DRIVER for WINDOWS
USER'S MANUAL

MCX & WAN-HDLC products ranges

ACKSYS Communications & Systems

www.acksys.fr
support@acksys.fr
sales@acksys.fr

March 2010 (revision A.4)
DTUS059

MCX and WAN-HDLC products ranges
Windows drivers

COPYRIGHT (©) ACKSYS 2010

This document contains information protected by Copyright.
The present document may not be wholly or partially reproduced, transcribed, stored in any computer or other system whatsoever, or translated into any language or computer language whatsoever without prior written consent from *ACKSYS Communications & Systems* - ZA Val Joyeux – 10, rue des Entrepreneurs - 78450 VILLEPREUX - FRANCE.

REGISTERED TRADEMARKS ®

- *ACKSYS* is a registered trademark of *ACKSYS*.
- Windows 7, Windows Vista, Windows XP, Windows NT, Windows 2000, Windows 2003 Server, MS-DOS, Windows 95 are registered trademarks of MICROSOFT.

NOTICE

ACKSYS ® gives no guarantee as to the content of the present document and takes no responsibility for the profitability or the suitability of the equipment for the requirements of the user.

ACKSYS ® will in no case be held responsible for any errors that may be contained in this document, nor for any damage, no matter how substantial, occasioned by the provision, operation or use of the equipment.

ACKSYS ® reserves the right to revise this document periodically or change its contents without notice.

ACKSYS COMMUNICATIONS & SYSTEMS
ZA Val Joyeux
10, rue des Entrepreneurs
78450 VILLEPREUX
FRANCE

Telephone:	+33 (0)1 30 56 46 46
Fax:	+33 (0)1 30 56 12 95
Web:	www.acksys.fr
Hotline:	support@acksys.fr
Sales:	sales@acksys.fr

TABLES OF CONTENTS

PRESENTATION 1

I TECHNICAL SPECIFICATIONS 2
I.1 General characteristics 2
I.2 Integration in Windows environment 3

II DOCUMENTATION 4

INSTALLATION 5

I BOARD OPERATING MODES 5

II PLUG & PLAY DRIVER INSTALLATION 7
II.1 Operating mode selection. 7
II.2 Physical board Installation 7
II.3 Reboot 7
II.4 Installed card setup 8

III WINDOWS NT DRIVER INSTALLATION. 14
III.1 Check the system configuration 14
III.2 Operating mode selection. 14
III.3 Board resources selection. 14
III.4 Physical board Installation 14
III.5 Reboot 14
III.6 "MCXSETUP" installation program 15

IV CHECKING THE INSTALLATION 24

V DEVELOPMENT TOOLS AND EXAMPLES 24

COM PORT COMPATIBILITY MODE 25

I APPLICATION PROGRAMMING INTERFACE (API) 25
I.1 Programming asynchronous communications 26
I.2 Programming synchronized asynchronous communications. 28
I.3 Programming synchronous HDLC/SDLC/BISYNC communications 29
I.4 Programming the LAPB (or HDLC-ABM) protocol 31
I.5 Programming driver-specific services. 33
I.6 Standard Windows utilities 34
I.7 Other utilities 35

II DETAILED REFERENCE MANUAL 36
II.1 Excerpt from the mcc_mcx.h file 36
II.2 SET/GET SYNC STATE functions 39

II.3 Example for SET_SYNC_STATE 42
II.4 CMD and CMD_AUTO functions....43
II.5 Examples for CMD and CMD_AUTO 46
II.6 ACCESS_AREA function 48
II.7 MCX_OPTIONS function 49
II.8 Examples for ACCESS_AREA and MCX_OPTIONS....51
II.9 Appendix : flow control. 53

III APPENDIX : SPECIFIC ERROR CODES. 54

IV APPENDIX : LIMITATION AND DIFFERENCES WITH THE COM PORTS 55

V COM PORTS FAQ 55

MCXDOS/AUTOMCX MODE 56

I DEVELOPMENT OF THE APPLICATION TO BE DOWNLOADED 56

II PROVIDED UTILITIES 56

III BOARD LOADING 56

IV LOW LEVEL PROGRAMMING INTERFACE 57

V KNOWN DEFECTS 58

VI USING MCXDEBUG.EXE 59

ANNEXES

GLOSSARY 63

PRESENTATION

This driver allows to use all functionalities of the boards in the MCX range (ISA, PCI 3V3, PCI 5V, cPCI buses).

Three versions of this driver exist :

- The Plug & Play version, for recent "Plug&Play" Windows systems, from Windows XP to Windows 7 (32 bits only), including Windows 2003.
- A separate Plug & Play version provided for Windows 2000. Its development was stopped at version 3.2.4 and it does not handle multicore processors.
- The older, « non Plug & Play » version for Windows NT 4.0.

It allows, either to download an application on the board, or to use the board through the Windows WIN32 COMM API. As such it is compatible with most applications written for this operating system and his successors (for example HyperTerminal).

This manual describes the version 3.2.8 of the Plug & Play driver, and the version 2.2.14 of the Windows NT driver, when used with the latest versions of the board firmwares. We try to maintain ascending compatibility between versions. If you are using an earlier version of the board, firmware or driver, some characteristics may depart from this documentation.

MCX range: A board consists of a **mother board** (according to the bus type) and a **daughter/extension board**.		
Bus	Mother board	Daughter board
3.3V & 5V PCI	MCXUNI	MCXBP MCXBPMR PCB/S PCB/U PCB/570 PCB/570-F2
5V PCI	MCXPCI	MCXBP MCXBPMR PCB/S PCB/U PCB/570 PCB/570-F2
cCPI 6U	MCXcPCI	MCXBP MCXBPMR PCB/S PCB/U PCB/570 PCB/570-F2
ISA	MCX	MCXBP ou MCXBPMR
ISA	MCX-Lite	Lite/U Lite/S Lite/570 Lite/104 Lite/485

In all this manual, the generic name MCX stands for any of these cards.

I TECHNICAL SPECIFICATIONS

I.1 General characteristics

Concerning supported baud rates, refer to GetCommProperties description (chapter « COM PORT COMPATIBILITY MODE ») and to the relevant firmware documentation (Basic software [DTUS014] or Multiprotocol software [DTUS016]).

Concerning available signals on the connectors, refer to the board hardware manual.

Plug & Play Driver for Windows

Operating systems	All 32-bits Windows starting with Windows 2000.
Board type	All except ISA boards
Number of boards	limited by the number of available slots
Basic software	version 2.7 or later
Multiprotocol Option	version 3.1 or later
Supported bus	PCI 3V, PCI 5V, CompactPCI 6U

Windows NT 4.0 driver

Attention, this driver also runs for the Windows Plug & Play operating systems, however it is not advised to use it because of the following reasons: In the device manager, it appears in the hidden and not Plug & Play device drivers and the COM ports are not listed and it shows a conflict the PCI device detected by Windows.

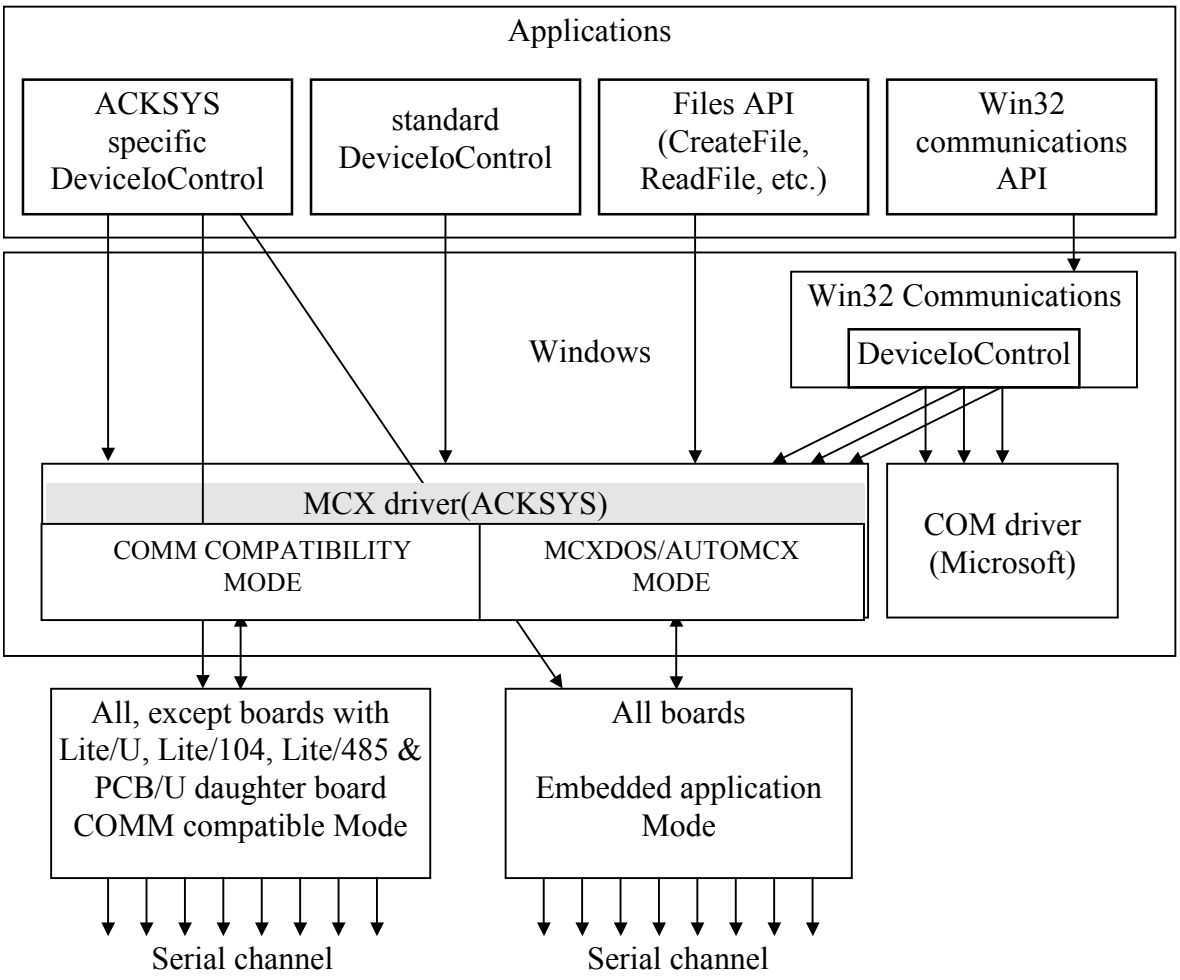
Operating systems	Windows NT 4 since SP3, single processor only.
Lmitations	Hyperthreading and multiprocessor not supported
Board type	All, ISA board included
Number of PCI/cPCI boards	limited by the number of available slots
Number of ISA boards	limited by available ISA bus resources (usually, 1 to 4 boards)
Basic software PCI/cPCI boards	version 2.7 or later
Basic software ISA board	version 2.0 or later
Basic software (MCC)	version 3.8 or later
Multiprotocol Option	version 3.1 or later
Supported bus	PCI 3V, PCI 5V, CompactPCI 6U, ISA

I.2 Integration in Windows environment

The driver allows:

- to access directly to board's resources (dual ported memory, FIFO, input output registers) in the case of specific applications embedded in the board.
- to use intelligent serial ports (asynchronous transmissions, synchronized asynchronous, synchronous bit oriented, synchronous character oriented), following interfaces defined in Win32 (COMM API) for traditional serial ports. Used in asynchronous mode, the driver tries to imitate as completely as possible the COM1 port driver provided by Microsoft. However, some differences remain, because so much processing takes place in the board.

The following diagram shows how the driver takes place in the Windows architecture :



The diagram shows that the driver uses the same API as the COM ports. If a specific application needs to use board features without accessing them via the Win32 API, it can access the driver or even the board directly via the DeviceIoControl commands described in the section entitled COM PORT COMPATIBILITY MODE and MCXDOS/AUTOMCX.

Typical examples of this kind of use are switching to RS422 or to synchronous mode.

II DOCUMENTATION

Various technical documentations come with the board, according to the options that you had chosen and how you will use you board. This manual sometimes refers to them.

Hardware descriptions :

These manuals describe, for each main board and extension board :

- physical and electrical characteristics
- connectors
- switches and jumpers
- internal peripherals programming.

EPROM firmware descriptions :

[DTUS014] Basic software user's manual
[DTUS016] Multiprotocol firmware user's manual

These manuals aimed to the system programmer, describe parameters and procedures used to manipulate onboard communication channels.

Drivers and development tools description :

[DTUS015] MCX range device drivers user manual (ISA bus, except Windows NT)
[DTUS004] MCXDOS user manual

These manuals describe how to use the board from an application on the host computer.

INSTALLATION

I BOARD OPERATING MODES

IMPORTANT : before using the board, you must chose the mode that you will use, according to your needs of your application.

Boards can be used in four differents modes.

- BASIC SOFTWARE Mode
  This mode allows an ascending compatibility with older boards (MCC-8, MCC-16 and MCC/II). It can drive up to 64 asynchronous serial channels with sensible performances.

- MULTIPROTOCOL Mode (optional)
  This mode offer a light compatibility with Basic software. It allows to drive from 2 to 64 asynchronous serial channels; moreover it allows to define synchronous protocols with an application interface similar to that of COM1/2. It takes in charge, not only boards supported by Basic software, but also board with Lite/570 or PCB/570 extension.

- MCXDOS/AUTOMCX Mode
  This mode allows, once an on-board DOS environment application has been developped, to download DOS and the board application from Windows. The development of the application, must be led on a DOS or compatible workstation (Windows 9x, etc.).

- BIOS Mode
  In this mode the board shall execute its BIOS extensions, which implies to execute either a specific application which must be fixed beforehand in its FLASH EPROM, or an operating system which is setup in its embedded hard disk.

Configuration.

The choice of operating mode is done on two levels : on-board switches and jumpers set the board's behaviour at power on or after a reboot (hence before any driver is started) ; "run mode" in the driver itself define how the board will interact with the application.

The hardware manual explains the settings of the switches and jumpers of the board.

On the PCI & cPCI boards, the jumpers and switches settings are shown on the transparent plastic sheet which protects the mezzanine card.

Mode	Bus	Mezzanine card	Switches and jumpers	Functionalities
Basic software	ISA	MCXBP MCXBPMR Lite/S	ST2/ST3 set to 1-2	COM1/2 Compatibility MCC Compatibility
	PCI	MCXBP MCXBPMR PCB/S	SW1 set to built-in. firmware	
	cCPI	MCXBP MCXBPMR PCB/S	JP2 set to built-in firmware	
Multiprotocol sotfware	ISA	MCXBP Lite/S Lite/570	ST2/ST3 pos. 1-2	COM1/2 Compatibility Synchronous protocols
	PCI	MCXBP MCXBPMR PCB/S PCB/570	SW1 set to built-in firmware	
	cPCI	MCXBP MCXBPMR PCB/S PCB/570	JP2 set to built-in. firmware	
Mcxdos/automcx	ISA	All	ST2/ST3 set to. 2-3	No COM1/2 compatibility Onboard execution of MS-DOS and specific application PC side interface to be specified by the application
	PCI	All	SW1 set to Mcxdos	
	cPCI	All	JP2 set to Mcxdos	
BIOS Extension	ISA	All	ST2/ST3 set to. 2-3	No COM1/2 compatibility Loads any operating system on board from its embedded disk Board boots at PC-side application software command.
	PCI	All	SW1 set to standalone	
	cPCI	All	JP2 set to standalone	

II PLUG & PLAY DRIVER INSTALLATION

Note that the dialogs shown are from Windows XP. Equivalent dialogs are used on Vista.

For Windows 2000 make sure you use the dedicated driver (newer versions won't install).

II.1 Operating mode selection.

Refer to section Board operating modes. WARNING: the Plug&Play driver doesn't handle ISA boards ranges.

II.2 Physical board Installation

WARNING: With Windows 2000, you must install the board in the computer after you run the installation program. With newer Windows versions, you must install the board in the computer before starting the installation program. Otherwise the program will not be able to recognize the board and allocate resources automatically.

II.3 Reboot

When restarting, Windows will detect a new device. The device installation wizard pops up: please ignore it (cancel it or let it alone, it will disappear after installation).

Insert the ACKSYS CD, browse it to find the driver directory. Run the MCXINSTALL.EXE program.

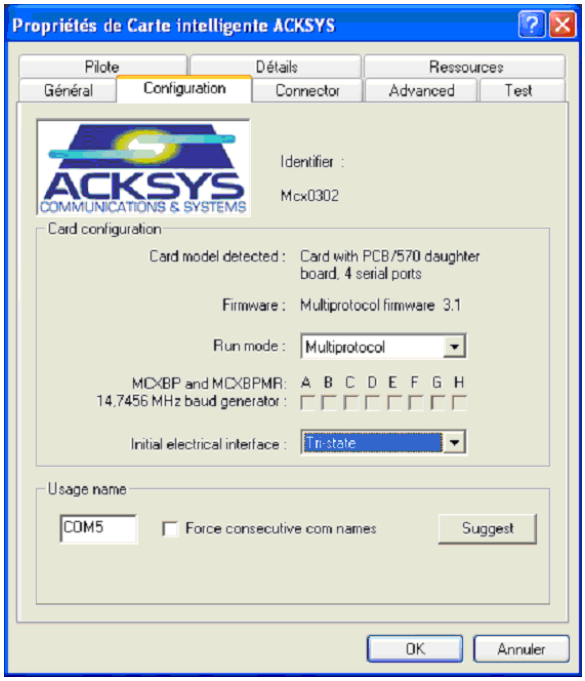
After installation, the board appears in the Device Manager in the "multiport serial" group. This icon allows disabling or configuring the board.

If the switches of the board are set to "built-in firmware" (which is the usual case), its ports will also appear in the "Ports (COM and LPT)" group.

II.4 Installed card setup

The properties dialog box has two specific tabs: “configuration” and “advanced”.

The **configuration** tab displays some information and allows basic setup.



Board basic properties

- **Model**

  Autodetected board model.

- **Run mode**

  Tells the driver the chosen operating mode (see section Board operating modes). Only operating modes supported by the device appear.
  Usually this combo is preset depending on the board switches position. You may however force another driver behavior for special applications (for instance, for firmware upgrade).
  If Multiprotocol mode is selected whereas the option is not setup in the EPROM board, during the boot, the pilot will save an error in the event viewer.

- **14,7456 MHz baud generator**

  On boards with MCXBP or MCXBPMR or PCB/S extension, each channel bloc has two oscillators for generation of transmission clocks. Only one of the two oscillators can be use at the same time. The oscillator which will be use must be specified at setup, according to precision wished onclocks calculation. It will be use by all channels on the same block. This precision can become significant in synchronous modes, or for high speed transfert (over 19200 bauds).

  Boards with MCXBP or MCXBPMR extension have one to eight blocks (A to H) of 8 channels each one. Boards with PCB/S extension get only one block (A) of 2 channels. When a box is checked, all

channels of corresponding block use the oscillator at 14,7456 MHz. Else, they use the oscillator at 16 MHz

For compatibility reasons the boards with PCB/570 extension handle these checkboxes though they have only one oscillator. They recompute the correct baud rates as if they where equipped with both oscillators.

- **Initial electrical interface**

  On some boards, the electrical interface can be preset at startup time. On these boards, when Multiprotocol option is enable, it is possible to indicate here which electrical interface the pilot must use on startup on all channels (before the user has an opportunity to set the electrical interface).

  WARNING: For PCB/570 mezzanines the default electrical interface is « high impedance » to avoid electrical hazard with connected devices. You must change the electrical interface before using the ports, using either this selection box or the provided API (described later).

  WARNING : this box has no effect with the MCX-BPMR extension.

- **Usage name**

  In MCXDOS/Automcx mode, this name will be used by application programs to access the board. In COM ports compatibility mode, the names are defined by Windows (warning, they are not necessarily in sequence).

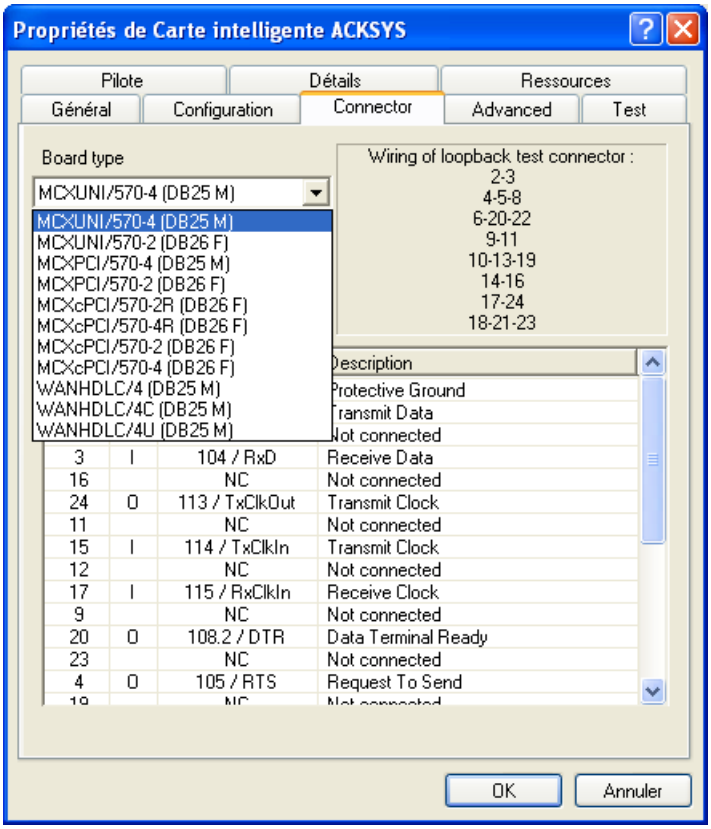
- **Force consecutive COM names**

  In COM ports compatibility mode, the port names will be reassigned with consecutive numbers starting from the one displayed. An error message shows when the provided name is invalid or when the span of numbers overlays an existing port (defined for another card).

- **Suggest**

  In COM ports compatibility mode, computes the first available COM port name. If "Force consecutive COM names" is checked, the corresponding constraint will be enforced while computing the name.

The **connector** tab describes the pinout and type of the DB connectors.

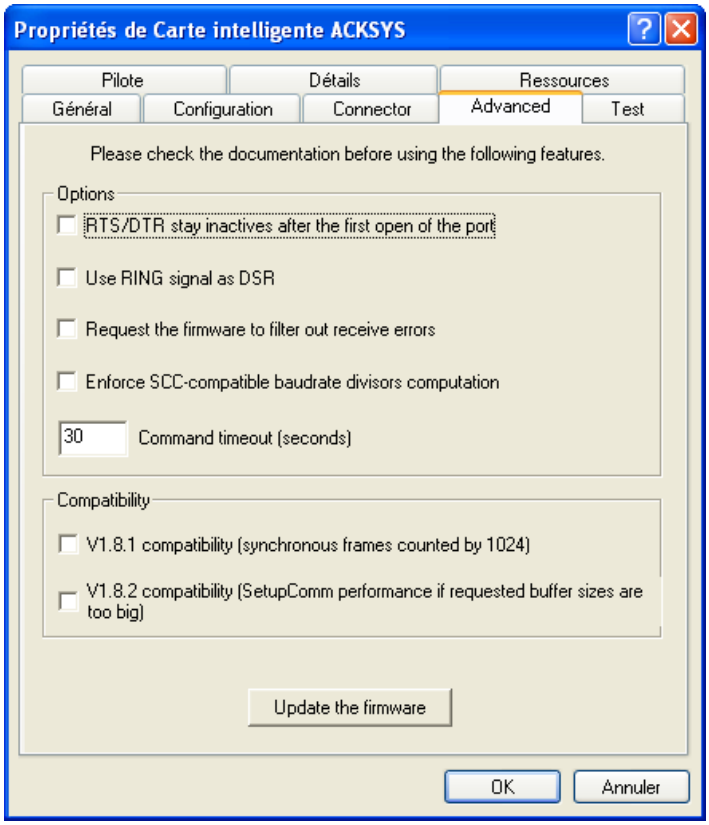


The **Avanced** tab allows to manage options and compatibility with earlier versions.

These options act on all board's channel.

Warning : for performances reasons, compatibility modes will not be supported indefinitely; hence it is strongly advised to upgrade existing software consequently.

Note : From the version 3.1.4 of the driver support for options related to compatibility with versions of the driver older than 1.8 are dropped; options related to ISA bus cards are dropped as well.



- **RTS/DTR stay low after the first CreateFile**

  Why : on startup, this driver, by compatibility with Microsoft™ serial port piote, assign RTS_CONTROL_ENABLE and DTR_CONTROL_ENABLE values to *fRtsControl* and *fDtrControl* fields of DCB structure. Consequently, first opening of a channel after driver startup, will automatically rise RTS and DTR signals. This behaviour can be annoying in some applications.

  Box checked: on startup, pilot assign to champs *fRtsControl* and *fDtrControl* fields of DCB structure values by default RTS_CONTROL_DISABLE and DTR_CONTROL_DISABLE. RTS and DTR signals will stay inactive until a voluntary change by the application.

- **Use RING signal as DSR**

  Why : some boards do not have the DSR pin on RS232 connector. Some applications use this signal.

  Box checked: the pilot inverts DSR and RING pins significance. The signal detected on the RING pin is provided to the application as DSR signal (and reciprocally for boards which have a DSR pin). Application's actions supposed to act on DSR act on RING signal instead (for example *fDsrSensitivity* modification).

  Notice : It is possible to chose this inversion separately for each channel (see section « Reference manual »).

- **Request the firmware to filter out receive errors**

  Why : some applications disable receive data errors signalling by clearing bit IT5 in the MINTR command. However in several occasions the driver automatically reinstates data error signalling.

  Box checked: the driver always clears bit IT5 in the MINTR command, thus effectively disabling receive data errors signalling in all operations.

- **Enforce SCC-compatible baudrate divisors computation**

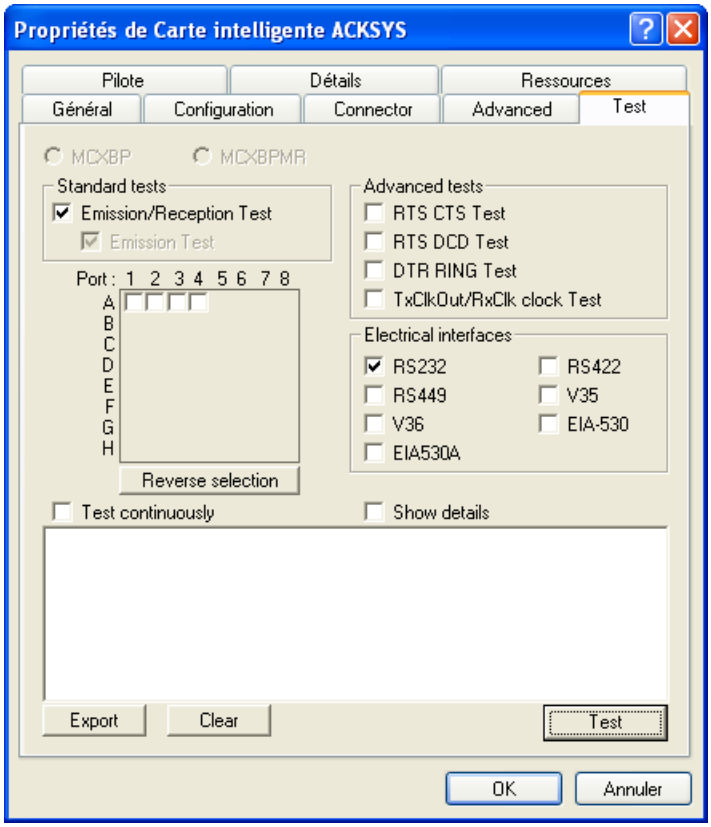
  Why : Recent versions of the firmware compute baudrate generators parameters by themselves from the requested baud rate. This computation was previously done by the driver, using a less sophisticated, but less precise, command. Hence baud rate computations done on cards equipped with differing firmware versions can incur lightly different results.

  Box checked: the driver always uses the oldest method (VINIT baud rate code 17).

- **Command timeout**

  To detect possible breakdowns, the driver checks the duration of certain command groups sent to the board. An error is signalled to the application, and a message is put in the System event viewer, if this duration exceed that the one indicated here. This error case is rare and the default value (30 seconds) should normally not be changed. Allowed values go from 5 to 3600 seconds.

The **Test** tab allows to test each channel.



III WINDOWS NT DRIVER INSTALLATION.

III.1 Check the system configuration

The driver must be installed on at least **Windows NT 4 Service Pack 3**. Check which service pack is installed. On previsous service packs, a kernel bug forbids the driver to start properly.

The driver does not work on multiprocessors, dual-core or Hyperthreading computers. Disable these modes in the computer BIOS if required.

III.2 Operating mode selection.

refer to section Board operating modes.

III.3 Board resources selection.

Needed resources for PCI or cPCI boards are automatically reserved. However you must check that switch SW1 matches the selected operating mode. For that defer to the board's user manual.

III.4 Physical board Installation

You must install the board in the computer before starting the installation program. Otherwise the program will not be able to recognize the board and allocate resources automatically.

III.5 Reboot

Windows NT 4.0 is not Plug & Play, hence it doesn't detect the board by itself. However some BIOSes quickly display the list of installed PCI boards, before starting th operating system. The board appears as a communication card, its VENDORID is 1528 and its DEVICEID is 0800.

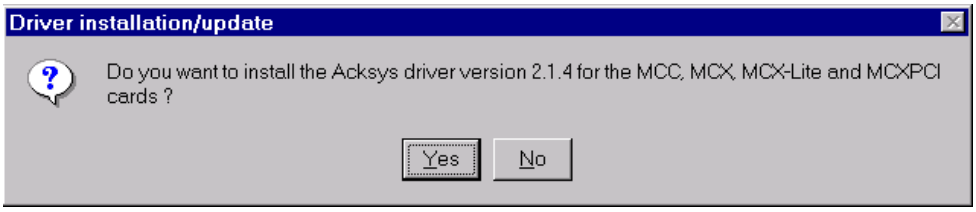
III.6 “MCXSETUP” installation program

Driver installation

When you have installed the board in the computer and restarted the operating system, place the browse the driver distribution medium to find the MCX Windows driver, and run the following command:

SETUP . BAT

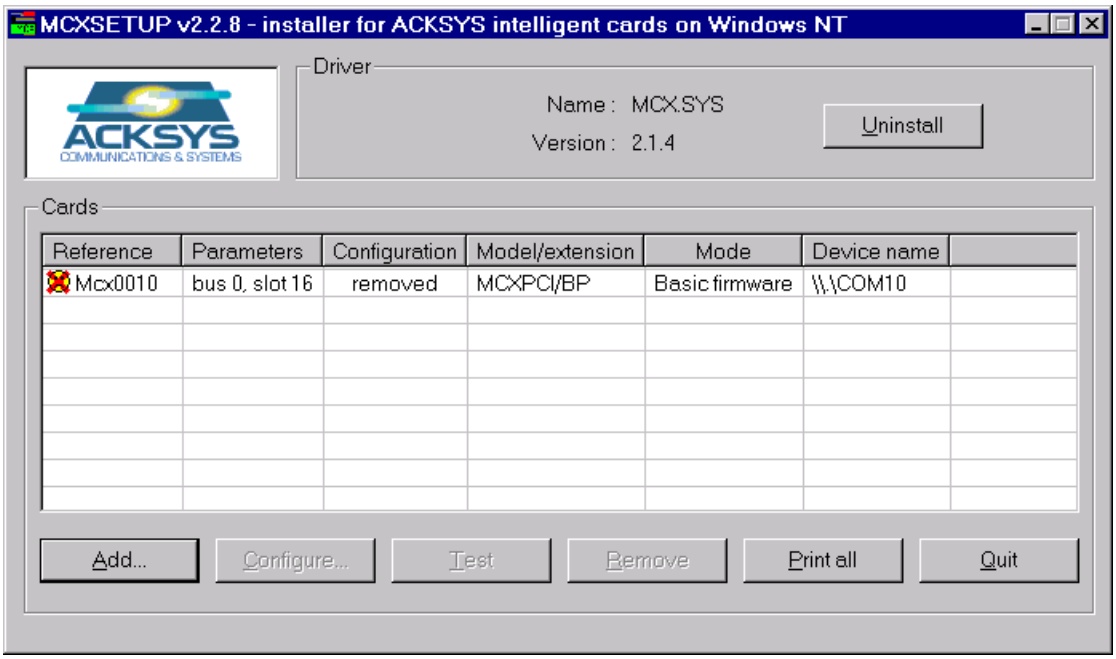
This script starts the installation and configuration program called “MCXSETUP”. The first time you execute the program, it will detect that the MCX.SYS driver has not yet been installed and ask you to confirm the installation :



If a previous version of the program is already installed on the disk, MCXSETUP checks that the version you want to install is more recent and that updating is possible, before proceeding with installation.

It then installs the “MCX.SYS” device driver and a number of utilities on the hard disk. You will then be able to run these utilities directly from the command line prompt, or from the menu bar (**start** → **execute** → **mcxsetup** → **OK**).

The first window displayed by MCXSETUP is split into two groups. The upper part lets you manage the driver installation process and the bottom part manages the installation and initialisation of the boards :



MCXSETUP main window

- **Version** indicates the version of the MCX.SYS driver file that is installed on your hard disk.
- **Uninstall** lets you delete MCX.SYS, MCXSETUP and the disk utilities, and deletes the driver information from the Registry.

Configuring installed boards

The list in the lower half of the window indicates the boards that have already been detected or installed and their main characteristics.

A red icon indicates that boards have been detected but that they are not yet configured and cannot be used. You must configure them before they can be used.

A yellow icon indicates that boards are inhibited, as a result of deliberate action (you have checked the "inhibit" box in the board properties). You can re-enable this boardin order to use it.

A green icon indicates useable boards have been recognized. Boards with this status can be used (provided that ISA resources are correct, in the case of ISA boards).

You can access a board by selecting it then using either of the buttons at the bottom of the window or clicking with the right-hand mouse button. Double-clicking opens the properties window (see below).

The Add… and Remove buttons let you define and delete an ISA board. The Configure… button opens the selected board's properties pages. The Test button lets you run some basic checks on the operation of a selected board, and displays a report after a few seconds. The Quit button checks that, after a board has been modified, the same resources are not being used by several boards, offers some installation advice, then ends the program.

The Print all button, accessible if no board has been selected, prints a summary of the configuration of each board. This button changes to Print when a board has been selected.

The overall configuration procedure is consequently as follows :

- **for PCI/cPCI board:**

1) Find the line that corresponds to the board you want to configure. If several PCI boards have been installed, they can be identified by their bus and slot numbers.
2) Double-click on the board to open the properties window.
3) Select the model from the list of boards.
4) Select operating mode according to the on-board switch position.
5) Select the other parameters according to your needs, and click on OK to close the properties window.
6) If operating mode is "Basic software" or "Multiprotocol", reselect the board in the list and click on Test.

- **For a ISA board:**

1) Click on Add… button.
2) Select the model from the list of boards.
3) Select operating mode according to jumpers position on the board.
4) In « Resources » tab, indicate resources choose according to jumper's position on the board.

   ISA boards resources allocation

   For each ISA board, you must select an interrupt line, an I/O port and a memory address area of 32 kilobytes. These resources must not be used by any other peripherals. Defer to [DTUS013] to set the board switches and jumpers according to your choices.
5) Select other parameters according to your needs, click on OK to close property window.
6) If operating mode is "Basic software" or "Multiprotocol", reselect the board in the list and click on Test.
7) Click on Quit. carefully read information messages which appear, and follow the instructions.

Notice on Plug-and-Play BIOS:

Recent micro-computers are equipped with “Plug-and-Play” capable BIOS. By default this kind of BIOS keeps for itself, all resources (Interruptions, memory address, Input/Output address) to allot to other boards respecting “Plug-and-Play” specifications.On SETUP BIOS screen, control that necessary resources are allotted to ISA bus. Control also the conflicts with other boards or integrated peripherals, like some mouse ports. Maybe you will need to defer to additional board’s manual or BIOS user’s guide.

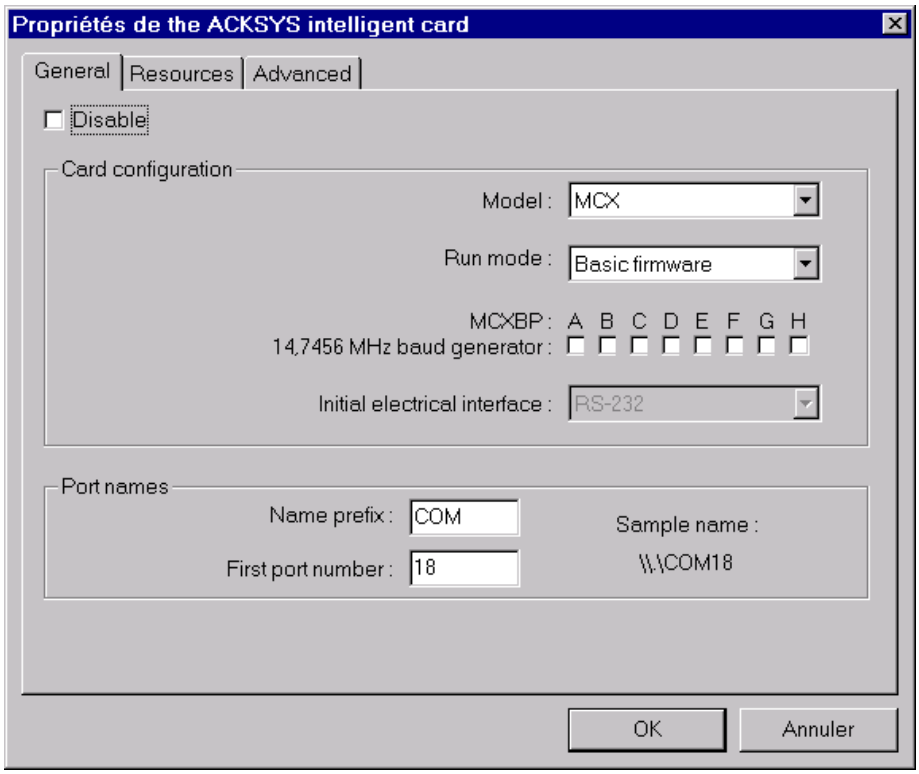
Installation and configuration of the driver

See section III.6. When the “MCXSETUP” installation program is started after installing a new ISA board, the board doesn’t appear on boards list, until you have configure it with Add… button.

Board properties

These parameters modify the operation of the device driver. They are analyzed at driver startup only. The driver can be restarted either when the system is rebooted, or manually from the "Devices" icon in the Windows Control Panel, or at the command line prompt by entering the "net stop mcx" command followed by "net start mcx", or by MCXSETUP with user confirmation.

Use the **General** tab to specify the characteristics of the board you want to install.



General properties of a board

- **Disable**

  Checking this box will prevent the driver from accessing the current board. You can use this to temporarily inhibit an installed board, or temporarily delete a board from the computer without generating messages in the Event Viewer.

- **Model**

  Selects the board model. Only models that correspond to the bus being used will be displayed in this list. The model must be specified accurately, as other options depend on your choice.

- **Run mode**

  Tells the driver the chosen operating mode (see section Board operating modes). Only operating modes supported by the device appear. If Multiprotocol mode is selected whereas the option is not setup in the EPROM board, during the boot, the pilot will save an error in the event viewer.

- **14,7456 MHz baud generator**

  On boards with MCXBP or MCXBPMR or PCB/S or Lite/S extension, each channel bloc has two oscillators for generation of transmission clocks. Only one of the two oscillators can be use at the same time. The oscillator which will be use must be specified at setup, according to precision wished onclocks calculation. It will be use by all channels on the same block. This precision can become significant in synchronous modes, or for high speed transfert (over 19200 bauds).

  Boards with MCXBP or MCXBPMR extension have one to eight blocks (A to H) of 8 channels each one. Boards with Lite/S or PCB/S get only one block (A) of 2 channels. When a box is notch, all channels of corresponding block use the oscillator at 14,7456 MHz. Else, they use the oscillator at 16 MHz

  For compatibility reasons the boards with PCB/570 or Lite/570 extension handle these checkboxes though they have only one oscillator. They recompute the correct baud rates as if they where equipped with both oscillators.

- **Initial electrical interface**

  On some boards, electrical interface is programmable. On these boards, when Multiprotocol option is enable, it is possible to indicate here which electrical interface the pilot must use on startup on all channels (before the user has an opportunity to set the electrical interface).

- **Name prefix**

  Name's prefixes which will be use to identify channels of this board. The string « COM » allows to use standard tools like HyperTerminal. Using an other name allow to get channel's name with a fixed format (see "CreateFile" description).

- **First port number**

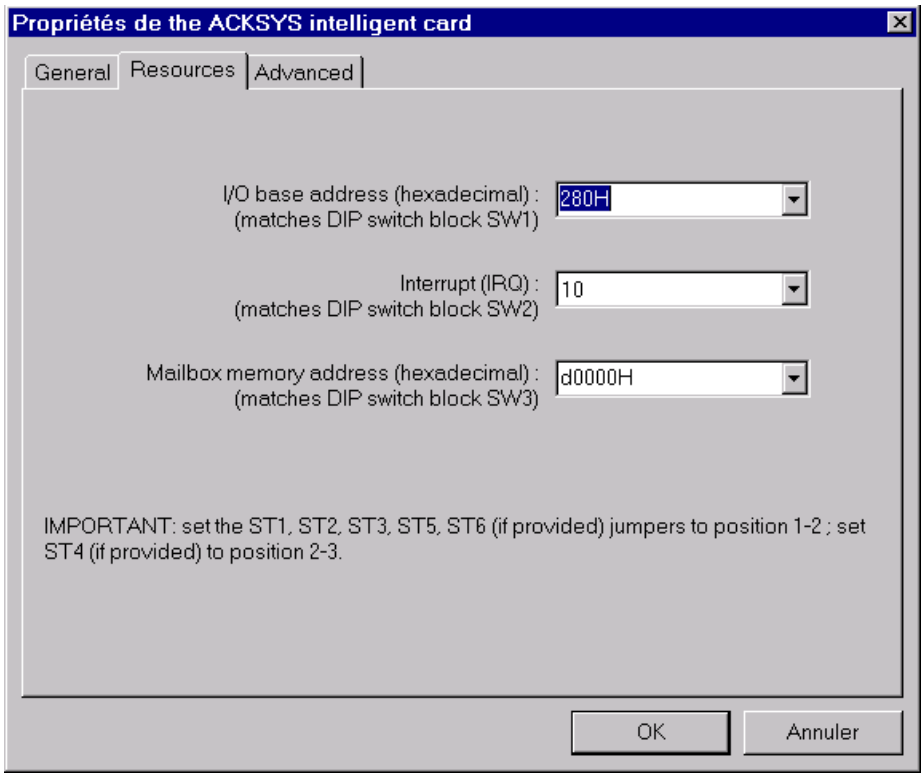
  This number will be assigned to the name of the board's first channel. The other channels will be numbered sequentially starting with this number.

- **Example**

  This displays the name of the board's first channel as it will be recognized by the driver. The channel names are formed by concatenating the fixed character string "\\.\", the name prefix and the number in the sequence starting with "first port number".

The Resources tab allows addresses and interruptions setup.

This tab is present only for board to ISA bus format. PCI board ressouces are allocated automatically.



resources tab for ISA board

- **I/O base address**

  Input-Output port of the board, used to reserve I/O resources in Windows, and to use boards which mailbox can be disabled.

- **Interrupt**

  interrupt request line chosen for the board. This interrupt must be affected to an ISA bus (by the PNP BIOS SETUP) and must be free (not used by an other peripheral). SW2 switches must reflect the chosen value.

- **Mailbox memory address**

  Address chosen for board's mailbox. A 32 Kb area is used starting from this address.

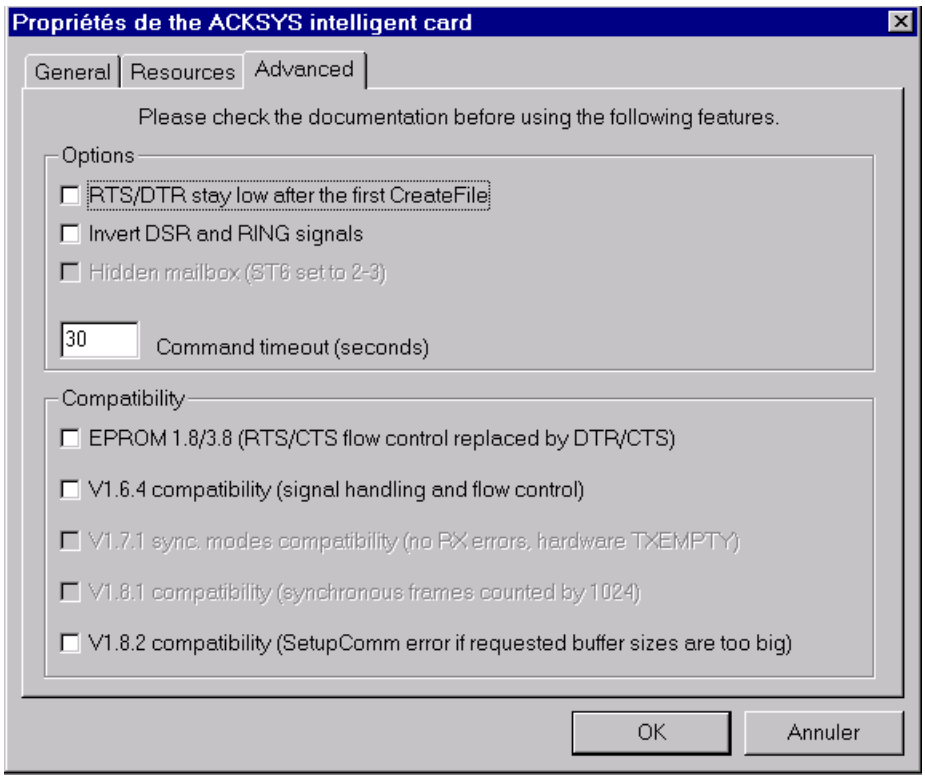
- **IMPORTANT :...**

  this note, in the bottom of the properties page, recalls you the correct board jumper position to allow proper communication with the driver.

Avanced tab allows to manage options and compatibility with earlier versions.

These options act on all board's channel.

Warning : for performances reasons, compatibility modes will not be supported indefinitely; hence it is strongly advised to upgrade existing software consequently.



- **RTS/DTR stay low after the first CreateFile**

  Why : on startup, this driver, by compatibility with Microsoft™ serial port piote, assign RTS_CONTROL_ENABLE and DTR_CONTROL_ENABLE values to *fRtsControl* and *fDtrControl* fields of DCB structure. Consequently, first opening of a channel after driver startup, will automatically rise RTS and DTR signals. This behaviour can be annoying in some applications.

  Box checked: on startup, pilot assign to champs *fRtsControl* and *fDtrControl* fields of DCB structure values by default RTS_CONTROL_DISABLE and DTR_CONTROL_DISABLE. RTS and DTR signals will stay inactive until a voluntary change by the application.

- **Invert DSR and RING signals**

  Why : some boards do not have the DSR pin on RS232 connector. Some applications use this signal.

  Box checked: the pilot inverts DSR and RING pins significance. The signal detected on the RING pin is provided to the application as DSR signal (and reciprocally for boards which have a DSR pin). Application's actions supposed to act on DSR act on RING signal instead (for example *fDsrSensitivity* modification).

  Notice : It is possible to chose this inversion separately for each channel (see section « Reference manual »).

- **Command timeout**

  To detect possible breakdowns, the driver checks the duration of certain command groups sent to the board. An error is signalled to the application, and a message is put in the System event viewer, if this duration exceed that the one indicated here. This error case is rare and the default value (30 seconds) should normally not be changed. Allowed values go from 5 to 3600 seconds.

IV CHECKING THE INSTALLATION

You can check that the driver has been properly started by consulting the Event Viewer, which displays two messages about the driver :

a) a message indicating that the driver has been loaded and its version,

b) for each board installed, a message indicates the version of the EPROM, the number of channels recognised by the board and other items of useful information.

In Windows 2000, Windows 2003 server, Windows Vista or Windows XP

You can check that the driver has been started correctly by consulting the Device Manager, accessible notably by right-clicking on the Workstation icon.

The options in the “Driver” tab let you stop, restart or inhibit the driver.

In Windows NT 4.0

You can check that the driver has been started by consulting the “Devices” icon in the Control Panel.

V DEVELOPMENT TOOLS AND EXAMPLES

You can communicate with the driver via your C language source programs, either through the standard functions of Win32 (e.g. CreateFile, etc.) or with the commands and structures that are specific to your driver.

The following files will be useful for developing your own applications :

Operating mode	Files
Basic software Multiprotocol	SDK\INCLUDE\MCC_MCX.H, SDK\INCLUDE\MCXPROTO.H
Mcxdos/Automcx BIOS Extension	SDK\AUTOMCX\INCLUDE\AUTONT.H, SDK\AUTOMCX\LIB\AUTOMCX.LIB

If necessary, copy these files on to your hard disk so that they can be accessed by your C language source programs.

You will find sample programs in the SDK\MULTIPROTOCOL subdirectory of the distribution medium for “Basic software” and “Multiprotocol” modes. More specifically, the SDK\MULTIPROTOCOL\LIB directory contains a library of functions that are useful for getting to know the driver; these functions use the “header” SDK\MULTIPROTOCOL \INCLUDE\ACK_W32.H and are compiled with the MultiThread option in the SDK\ MULTIPROTOCOL\LIB\MCC_MCX.LIB. library.

COM PORT COMPATIBILITY MODE

The following sections describe how the driver operates with the MCX MULTIPROTOCOL on-board firmware.

I APPLICATION PROGRAMMING INTERFACE (API)

Each serial channel can be programmed independently, whether as regards the electrical interface, the signal format, the frame format, the protocol or the flow control, etc. Each channel can be driven by a different application if necessary.

The file name used to access the channels is defined when the board is configured. The "usage name" is "COM" followed by a number which is defined in the general configuration tab of the card.

NOTICE: The Windows driver can also handle "usage names" other than "COM". In this case the channels will be numbered 01, 02, etc. up to the number of channels installed on the board.

Examples

Usual name	First port number	Channel identification
COM	3	`\\.\COM3`, `\\.\COM4`, etc.
COM	9	`\\.\COM9`, `\\.\COM10`, etc.
Mcx1 (allowed in NT4 only)	Ignored	`\\.\Mcx101`, `\\.\Mcx102`, etc.
McxB (allowed in NT4 only)	Ignored	`\\.\McxB01`, etc.

The available APIs are based on the Win32 documentation and fall into three groups : **file services** (CreateFile, ReadFile, etc.), **communication services** (SetCommState, etc.) and the **DeviceIoControl** file service which is used by all the driver-specific functions that are not provided in Win32 (change of protocol or electrical interface, etc.).

In the following pages, the APIs are presented according to whether they are used in :

- ☞ asynchronous communications,
- ☞ synchronous communications,
- ☞ LAPB communications,
- ☞ driver-specific services,
- ☞ utilities.

I.1 Programming asynchronous communications

For general information, consult Microsoft's Win32 documentation. The details below simply concern :

- ☞ cases in which the driver differs from Win32 serial communications specifications,
- ☞ cases in which the Win32 documentation is vague,
- ☞ cases where the specification is not easily understood.

File services

Service	Comments
CreateFile	after **CreateFile**, the transmit and receive buffers are empty.
WriteFile	the number of outgoing characters must be less than or equal to 8,192 bytes. **WriteFile** ends when the outgoing characters are in the board buffer, and not when they are actually transmitted.
CloseHandle	**CloseHandle** ends by deleting the transmit buffer and opening circuits 105 and 108 (RTS and DTR). It is advisable to use **FlushFileBuffers** before **CloseHandle** to ensure that all the data in the buffer has been transmitted.

Serial communication services

Service	Comments
BuildCommDCB	*unrestricted.*
BuildCommDCBAndTimeouts	Same comments as for SetCommTimeout.
ClearCommBreak	No effect, the BREAK ends automatically after one second.
ClearCommError	Only the CE_BREAK, CE_FRAME, CE_OVERRUN, CE_RXPARITY bits in lpdwErrors are supported. In the COMSTAT structure, only the cbInQue and cbOutQue fields are supported.
EscapeCommFunction	CLRDTR, CLRRTS, SETDTR, SETRTS, SETBREAK are supported. CLRBREAK : see ClearCommBreak. SETXON, SETXOFF are not supported.
GetCommMask	*unrestricted.*
GetCommModemStatus	see "Appendix : limitations and differences with the COM ports.", page 55.

GetCommProperties	*dwMaxBaud* : BAUD_USER, because any speed supported to within 1% is accepted. *dwProvSubType* : RS232, even though these boards also support the RS422, RS485, current loop types if they have the necessary options. *dwMaxTxQueue, dwMaxRxQueue* : total number of bytes effectively allocated to the channel buffers. Support for the other fields is standard.
GetCommState	Returns the values specified for SetCommState (except fParity).
SetCommState	The value RTS_CONTROL_TOGGLE for fRtsControl, and the fBinary and EofChar fields are not supported. With the Multiprotocol options, the other fields are supported normally. With the basic software, the driver also ignores the fields below and uses the fixed value indicated :

fDsrSensitivity	FALSE
fTXContinuesOnXoff	TRUE
fErrorChar	FALSE
fNull	FALSE
XonLim, XoffLim	128, 20
fInX, *fOutX, fOutxDsrFlow*, *fOutxCtsFlow*, *fDtrControl, fRtsControl*	limited support : see "Appendix : flow control.", page 53.

GetCommTimeouts	*unrestricted.*
PurgeComm	*unrestricted.*
SetCommBreak	The duration of the BREAK is predefined (set to 1 second).
SetCommMask	For EV_RING and EV_DSR, see "Appendix : limitations and differences with the COM", page 55. EV_RX80FULL does not produce the expected results and EV_RXCHAR can be 100 ms late relative to the event.
SetCommTimeouts	*ReadIntervalTimeout* is supported with an error of 25% + 100 ms.
SetupComm	Non-standard buffer sizes are ignored. An error may be indicated in some cases (see the section on the configuration of advanced properties, page 15). The standard sizes are **8,192** characters per channel for **transmit buffers**, and **512** characters per channel for **receive buffers.**
TransmitCommChar	Not supported.
WaitCommEvent	*unrestricted.*

I.2 Programming synchronized asynchronous communications.

For general information, consult Microsoft™'s Win32 communications documentation. The main difference with an asynchronous port, comes from the required clock handling, with can be set with the following API call :

DeviceIoControl(...,IOCTL_SERIAL_SET_SYNC_STATE,...)

Sets the protocol (synchronous, asynchronous, synchronized asynchronous) and the associated options. For a full description, see page 39.

I.3 Programming synchronous HDLC/SDLC/BISYNC communications

For general information, consult Microsoft™'s Win32 communications documentation. The information below simply discusses :

- ☞ cases where the Win32 serial communication API is inappropriate,
- ☞ cases where the driver differs from the Win32 specifications,
- ☞ cases in which the Win32 documentation is vague.

File services

Service	Comments
CreateFile	after CreateFile, the transmit and receive buffers are empty.
ReadFile	**ReadFile** always returns a maximum of one frame. If the frame exceeds the requested length, only the first part will be returned and the rest lost. If the frame is shorter than the requested length it is returned in full and the requested buffer is not completely filled (even if another frame is already present in the board buffer : it will be provided with the next **ReadFile**). If the frame contains an error (ABORT, CRC, etc.) **ReadFile** returns an error (see also the compatibility options in the section on installation).
WriteFile	each **WriteFile** creates a frame whose size is limited to the value specified by the IOCTL_SERIAL_SET_SYNC_STATE command. **WriteFile** ends as soon as the outgoing characters are no longer in the board's buffer and not when they are effectively transmitted.
CloseHandle	**CloseHandle** ends by deleting the transmit buffer and opening circuits 105 and 108 (RTS and DTR). It is advisable to use **FlushFileBuffers** before **CloseHandle** to ensure that all the data in the buffer has been transmitted.
FlushFileBuffers	**FlushFileBuffers** ends with the transmission of the flag of the last transmitted frame (by the immediately preceding **WriteFile**). This guarantees that the board's buffers are empty.

Serial communication services

Service	Comments
BuildCommDCB	*unrestricted.*
BuildCommDCBAndTimeouts	See the comments for SetCommTimeout.
ClearCommBreak	No effect. The BREAK ends automatically after one second.
ClearCommError	Only the CE_BREAK, CE_FRAME, CE_OVERRUN, and RXPARITY bits in lpdwErrors are supported. In the COMSTAT structure, only the cbInQue and cbOutQue fields are supported; they specify a number of frames and not a number of characters.

EscapeCommFunction	CLRDTR, CLRRTS, SETDTR, SETRTS, SETBREAK are supported. CLRBREAK : see ClearCommBreak(). SETXON and SETXOFF are not supported.
GetCommMask	*unrestricted.*
GetCommModemStatus	see "Appendix : limitations and differences with the COM ports.", page 55.
GetCommProperties	*dwMaxBaud* : BAUD_USER, because any speed supported to within 1% is accepted. *DwProvSubType* : RS232, even though these boards also support the RS422, RS485, current loop types if they have the necessary options. *dwMaxTxQueue, dwMaxRxQueue* : total number of frames effectively allocated to the channel buffers. Support for the other fields is standard.
GetCommState	Returns the values specified for SetCommState *(except fParity).*
SetCommState	The only fields supported are *BaudRate, fOutxCtsFlow, fOutxDsrFlow, fDsrSensitivity, fDtrControl, fRtsControl, ByteSize* and *Parity* (the last two only in some protocols).
GetCommTimeouts	*unrestricted.*
PurgeComm	*unrestricted.*
SetCommBreak	The duration of the BREAK is predefined (set to 1 second).
SetCommMask	The EV_RXFLAG event is not supported. For EV_RING and EV_DSR, see "Appendix : limitations and differences with the COM ports. ", page 55.
SetCommTimeouts	*ReadIntervalTimeout* is ignored, except the value MAXDWORD. Otherwise, it should be initialised to zero to prevent any future incompatibility.
SetupComm	No effect.
TransmitCommChar	Not supported.
WaitCommEvent	*unrestricted*. EV_RXCHAR signals the arrival a complete incoming frame.

Driver-specific services

Service	**Comments**
DeviceIoControl(...,IOCTL_SERIAL_SET_SYNC_STATE,...)	Sets the protocol (synchronous, asynchronous, synchronized asynchronous) and the associated options. For a full description, see page 39.

I.4 Programming the LAPB (or HDLC-ABM) protocol

In addition to the specific details provided below on the programming of a LAPB connection, see the previous section, describing synchronous connections in general.

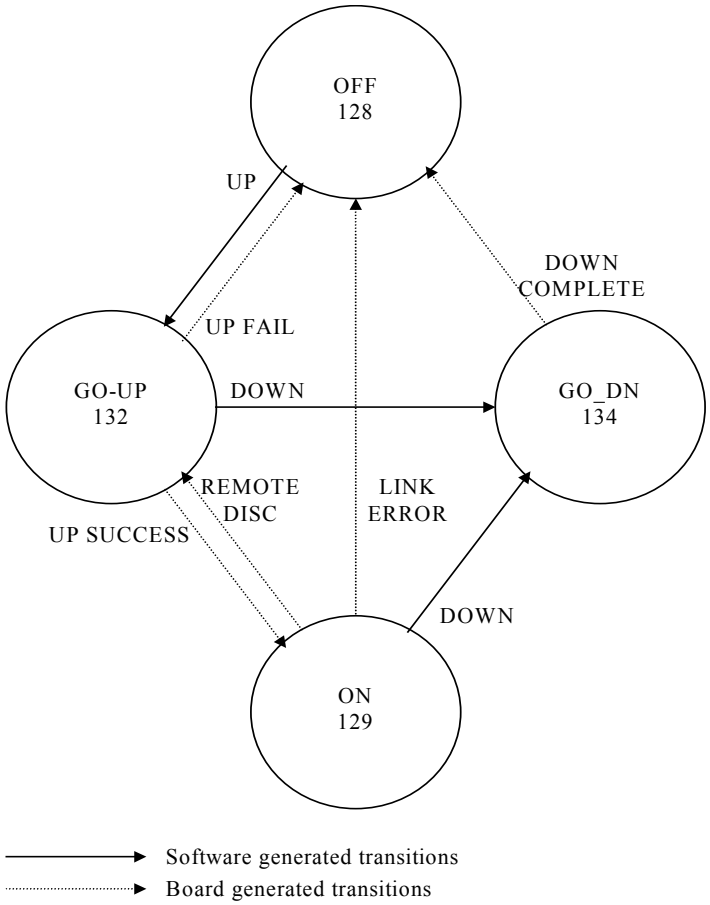
The informations below are necessary to program a LAPB link. Note that the **MCXMODE** utility has some limited capabilities for managing the Data Link layer. These can be used, in particular, in the test phase. See the directory **sdk\multiprotocol\LIB\src** for programming samples.

Primitives linked to the use of LAPB

Service	**Comments**
CreateFile	Keeps the current protocol, the electrical interface and the transmission speed. Reactivates the signal management mode. Purges the receive buffer. **CreateFile** does not attempt to create the connection.
CloseHandle	Purges the transmit buffer. In LAPB mode, the receiver is not inhibited. Protocol frames can thus be received and answered. **CloseHandle** does not attempt to break the connection.
ReadFile	Reads a frame in the board buffer or awaits the arrival of an information frame. Returns ERROR_HANDLE_EOF if the link is disconnected or broken during the wait. **ReadFile** is subject to the timeouts set by **SetCommTimeouts**.
WriteFile	Places a frame in the transmit buffer and waits if the buffer is full. Returns ERROR_NOT_READY if the link is disconnected or broken during the wait. **WriteFile** is subject to the timeouts set by **SetCommTimeouts**.
SetCommMask/GetCommMask/WaitCommEvent	The EV_MCXLAPB event (the equivalent of EV_EVENT1 in Win32) can be processed. It is triggered each time an interrupt of the LAPB state is received (see list of states below).
DeviceIoControl(...,IOCTL_SERIAL_SET_SYNC_STATE,...)	Sets the protocol and the associated options. See the “Reference Manual” section.
DeviceIoControl(...,IOCTL_SERIAL_CMD_AUTO,...)	Sends a command not directly supported by the Win32 interface. . In particular, the **PRCTL** command enables the state of the protocol to be checked with the ABMLINKUP, ABMLINKDN, and ABMSTATE options. See the sample files and the detailed reference in section “CMD and CMD_AUTO functions”.

Driver states

A LAPB channel can be in one of the four states shown in the diagram below :



New state	Triggered by
MCX_LINK_OFF (down, disconnected)	• DeviceIoControl(SET_SYNC_STATE) switching to LAPB protocol • DeviceIoControl(CMD_AUTO) with PRCTL ABMLINKDN command (if the disconnection occurs after the command has finished) • State interrupt, failure of an attempted connection by PRCTL ABMLINKUP • State interrupt, disconnection following a fatal transmission error • State interrupt, successful disconnection request by PRCTL ABMLINKDN
MCX_LINK_GO_UP (connection in progress)	• DeviceIoControl(CMD_AUTO) with PRCTL ABMLINKUP command • State interrupt, temporary disruption following a recoverable error
MCX_LINK_ON (up, connected)	• DeviceIoControl(CMD_AUTO) with PRCTL ABMLINKUP command (if the connection occurs before the end of the command) • State interrupt, reception of a request or acknowledgement of a connection
MCX_LINK_GO_DN (break in progress)	• DeviceIoControl(CMD_AUTO) with PRCTL ABMLINKDN command

Effect of state changes

No action is taken when the state change is caused by the application. The table below summarises the action taken when a state change is interrupted.

New state	Action
MCX_LINK_OFF	EV_MCXLAPB signalled if specified by an earlier SetCommMask Any ReadFile will return 0 characters and the ERROR_HANDLE_EOF error Any WriteFile will return the ERROR_NOT_READY error.
MCX_LINK_ON	EV_MCXLAPB signalled if specified by an earlier SetCommMask

Current status information

There are three ways to obtain information on the state of the connection.

- The first, imprecise, consists of assuming that the PRCTL/ABMLINKUP command always switches to the MCX_LINK_ON state, that PRCTL/ABMLINKDN always switches to the MCX_LINK_OFF state, and detecting any disconnections that may occur by analysing the errors returned by ReadFile and WriteFile. With this method a LAPB channel can be used as a filter with the system's standard commands, by establishing and breaking the connection with the MCXMODE utility.
- The second method consists of using the PRCTL/ABMSTATE to query the channel.
- The third method consists of using WaitCommEvent to instantly detect state changes that have been caused by the board or by the other end of the connection. As a rapid sequence of events can result in just one wake-up of WaitCommEvent, it is advisable to systematically query the channel state after a wake-up, and to consider that some steps in the sequence of states can be ignored. Note that the events are memorised as soon as they are selected by SetCommMask, and that, subsequently, one or more events occurring between two consecutive WaitCommEvent services cannot be lost (but may be combined into one event).

All three sources of information should be used in a complete process.

I.5 Programming driver-specific services.

The Win32 **DeviceIoControl()** function has been extended to dialogue directly with the software installed on the board. This makes it possible to send the commands manually, i.e. bypassing the driver. For details of these commands, see the "MCX board basic software user manual" and the "MCX board multiprotocol software user manual". For a full description, see page 41, and for an example of how to use the command, see the program sources in "sdk\multiprotocol\examples".

I.6 Standard Windows utilities

If you just want to use the standard Windows utilities with your boards, it isadvisable to select the name “COM” in the installation procedure.

Control panel

Do not use the “ports” icon to create the COM channels for boards because the driver creates these names automatically. However, you can use this dialogue to change the transmission parameters (this is useful if a channel is used by the Print Manager). Moreover, you should not try to use this dialogue to define addresses or interrupts.

HyperTerminal.exe

This Windows NT 4 accessory works normally with this driver.

Mode.exe

This DOS utility only supports the names that begin with COM. It can only be used to modify asynchronous transmission modes. Our **mcxmode.exe** utility can be used to work around these restrictions (see the description of **mcxmode.exe**).

Remote Access Services

The **Windows NT 4.0** dialling interface does not work with this driver.

Serial printers

Because of a defect in the Windows NT 3.51 Print Manager, you should proceed as follows to create a printer :

- Use the “Ports” icon in the “Control Panel” to modify or at least display the parameters for the particular port. You must use the <OK> button to validate.
- Use the “Services” icon in the “Control Panel”: stop the service called “Spooler” and restart it immediately. This will recognise the new ports and its parameters.
- “Printers” icon in the “Control Panel”: in the Print Manager, go to the “Printer” menu and select <Create a printer...>. Fill in the form; the port should appear in the <Print to:> list. The installation procedure ends here.

Command.exe (command line prompt in the DOS window)

The names COM1 thru COM9 can be used directly here. Otherwise, use the standard names \\.\name-assigned-on-installation, for example, to redirect console output :

`C:> DIR > COM9`	} Either format may
`C:> DIR > \\.\COM9`	} be used
`C:> DIR > \\.\COM10`	This is the only format authorised if the installed name is COM
`C:> DIR > \\.\MCX101`	This is the only format authorised if the name is MCX1

I.7 Acksys extra utilities

MCXSTARTER, available in the start menu, allows you to get an overview of the card capabilities, to test the appropriate configuration for your system and to generate initialization code.

The **MCXMODE** utility can be used to customise the transmission characteristics in synchronous or asynchronous mode. The MCXMODE command, used without parameters, displays on-line help.

The **MCCIOCTL** utility can be used to send the command as an argument directly to the board. The MCCIOCTL command, entered without parameters, displays on-line help.

The **DOSDEV** utility manages the links between the names of Windows NT objects and DOS peripherals. It remains effective until the system is shut down. The syntax is as follows :

`dosdev`	lists the aliases.
`dosdev -l COMn \Device\McxCNN`	creates the COMn alias for the McxCNN device.
`dosdev -r COMn`	deletes the COMn alias.

(where n is the number of the COM port, C is the number of the MCX board and NN is the number of the channel on the board).

The **SETMCX** utility can be used to display or modify a board's parameters from the command line or in a "batch" process :

`SETMCX`	on-line help
`SETMCX` *`n`*	displays all the parameters of board *n*
`SETMCX` *`n param`*	displays the value of the *param* parameter for board *n*
`SETMCX` *`n param val`*	assigns the value *val* to the *param* parameter for board *n*
`SETMCX` *`n`* `Compatibility +`*`code –code`*...	adds/removes the *code* bit to/from the configuration options

II DETAILED REFERENCE MANUAL

The driver-specific functions can be accessed via **DeviceIoControl()**. They use definitions and structures described in "**mcc_mcx.h**" or in "**mcxproto.h**" for functions that are specific to Multiprotocol mode.

These files are provided on the distribution medium in the directory "SDK\MULTIPROTOCOL\INCLUDE".

II.1 Excerpt from the mcc_mcx.h file

#include "mcc_mcx.h"

```
/* serial IOCTL codes for Windows NT */
#if defined(CTL_CODE) && defined(FILE_DEVICE_SERIAL_PORT)

#define MCX_IOCTL(code) \
             CTL_CODE(FILE_DEVICE_SERIAL_PORT,code,\
                  METHOD_BUFFERED,FILE_ANY_ACCESS)
#define IOCTL_SERIAL_GET_SYNC_STATE    MCX_IOCTL(0x901)
#define IOCTL_SERIAL_SET_SYNC_STATE    MCX_IOCTL(0x902)
#define IOCTL_SERIAL_CMD               MCX_IOCTL(0x903)
#define IOCTL_SERIAL_CMD_AUTO          MCX_IOCTL(0x904)
#define IOCTL_SERIAL_ACCESS_AREA       MCX_IOCTL(0x90A)
#define IOCTL_SERIAL_MCX_OPTIONS       MCX_IOCTL(0x90B)

#endif /* Windows NT */

------------------------------------------------------------

/* macros and structs for CMD & CMD_AUTO */
typedef struct mcc_cmd {
       unsigned char opcode;
       unsigned char status;
       unsigned char par[76];
       unsigned char ichan;
       unsigned char icond;
       unsigned char ipar1;
       unsigned char ipar2;
       unsigned char ipar3;
       unsigned char padding1;
       unsigned char *data;
       unsigned char *kdata;
       unsigned short length;
       unsigned short padding2;
}mcc_cmd;

typedef struct _MCC_CMD {    /* buffer for ioctls CMD... */
       mcc_cmd Cb;
       unchar Data[1];
} MCC_CMD, *PMCC_CMD;

/* size assigned to contain the struct _MCC_CMD */
#define MCX_DIRECT_IO_BUFFER_SIZE(datalen)
       ((datalen)+sizeof(mcc_cmd))
```

```
typedef struct _MCX_SYNCHRONIZATION_PARAMETERS {
 unchar SynchronousMode;          /* protocol */
#define MCX_SYNC_CHAR       0    /* NOT synchronous */
#define MCX_SYNC_BISYNC     2
#define MCX_SYNC_HDLC       4
#define MCX_SYNC_LAPB       5    /* LAPB, HDLC/ABM */
 unchar Duplex;                   /* flag version+2 bits duplex */
#define MCX_WAY_VERSION     0x80 /* validity bit Version field */
#define MCX_WAY_FULLDUPLEX  0x80 /* full duplex + version */
#define MCX_WAY_HALFDUPLEX  0x81 /* low RTS in each frame */
                                 /*--------- horloges --------- */
                                 /*   ETTD      ETCD NULL-MODEM */
 unchar TransmitClockSource;     /*   TXCI      BRG     BRG     */
 unchar ReceiveClockSource;      /*   RXC       BRG     RXC     */
 unchar TxClockPinSource;        /*  TRXC_HIGH  BRG     BRG     */
#define MCX_CLOCK_RXC       0    /* modes pour TransmitClockSource... */
#define MCX_CLOCK_TXCI      1    /* ... et ReceiveClockSource */
#define MCX_CLOCK_TRXC_HIGH 0    /* TxClockPinSource = always high */
#define MCX_CLOCK_TXCLOCK   1    /*     = copy of  TransmitClockSource */
#define MCX_CLOCK_BRG       2    /* common  modes */
#define MCX_CLOCK_DPLL      3
 unchar MonosyncChar;
 unchar BisyncChar;
 unchar Version;                  /* valid if Duplex = MCX_WAY... */
 McxUnshort Options;              /* default: 0 */
#define MCX_HDLC_USERDTR    1    /* slow HDLC but enable DTR */
#define MCX_HDLC_SPECS      2    /* Use the Protocol .Hdlc struct below */
#define MCX_BISYNC_SPECS    2    /* Use the Protocol . Bisync struct below */
 McxUnshort DataLength;           /* max. frame size (LAPB N1) */
#define MCX_FRAMELEN_DEFAULT 0   /* default for above field */
union{
    struct{
         unchar RxFrames;        /* default: 14 */
         unchar TxFrames;        /* default: 4 */
         McxUnshort Spare1;      /* reserved, set to zero */
         McxUnshort Spare2;      /* reserved, set to zero */
         McxUnshort Spare3;      /* reserved, set to zero */
         McxUnshort Spare4;      /* reserved, set to zero */
         McxUnshort Spare5;      /* reserved, set to zero */
    }Hdlc, Bisync;
#define MCX_HDLC_DEFAULT    0    /* default for this fields */
    struct{
         unchar Role;            /* default: CLIENT */
#define MCX_ROLE_CLIENT     1
#define MCX_ROLE_NETWORK    3
         unchar K;               /* default: 7                    */
         McxUnshort N2;          /* default: 10 essais            */
         McxUnshort T1;          /* default: 2550 ms              */
         McxUnshort T2;          /* default: 0                    */
         McxUnshort T3;          /* default: infini               */
         McxUnshort Spare;       /* reserved, set to zero         */
#define MCX_LAPB_DEFAULT    0    /* default for theses fields     */
    }Lapb;
 }Protocol;
} MCX_SYNCHRONIZATION_PARAMETERS, *PMCX_SYNCHRONIZATION_PARAMETERS;
```

```
/* structure pour IOCTL_SERIAL_ACCESS_AREA */
typedef struct _MCX_AREA_DESCRIPTOR {
    long Operation;             /* combin.of the following flags */
#define MCX_AREA_GET       0    /* board to application */
#define MCX_AREA_SET       1    /* application to board */
#define MCX_AREA_MEMORY    0    /* acces to mailbox */
    long StartAddress;          /* starting address of access */
                                /* relative to the base port or */
                                /* the start of the mailbox */
    long Length;                /* lenght to transfer */
    unchar Buffer[1];           /* values to write if MCX_AREA_SET */
} MCX_AREA_DESCRIPTOR, *PMCX_AREA_DESCRIPTOR;
#define MCX_AREA_DESCRIPTOR_SIZE(dlen) \
                           ((dlen)+sizeof(MCX_AREA_DESCRIPTOR)-1)

-------------------------------------------------------------------

/* structure for IOCTL_SERIAL_SET_OPTIONS */
typedef struct _MCX_OPTION {
    long Option;                                /* option code */
#define MCX_OPTION_GET_CHANNEL     0x20002 /* get channel n°  */
#define MCX_OPTION_SET_DSR_RI_INVERSION 0x30100 /* exchange DSR/RING */
#define MCX_OPTION_GET_DSR_RI_INVERSION 0x40001 /* get DSR/RING state */
#define MCX_OPTION_GET_COMPATIBILITY    0x50004 /* get current Compatibility */
    union{
        long Long[1];
        short Short[1];
        unchar Char[1];
    }Value;                                     /* parameters used by option */
} MCX_OPTION, *PMCX_OPTION;
#define MCX_OPTION_SIZE(dlen) \
                           ((dlen)+sizeof(MCX_OPTION)-sizeof(long))
```

II.2 SET/GET SYNC STATE functions

Two functions have been added to customise the format of synchronous frames.

#include "windows.h"
#include "winioctl.h"
#include "mcc_mcx.h"

DeviceIoControl(*hDevice*, **IOCTL_SERIAL_SET_SYNC_STATE**,
frameFormatBuffer, **sizeof(MCX_SYNCHRONIZATION_PARAMETERS)**,
NULL, **0**, *lpcbBytesReturned*, *lpoOverlapped*)

DeviceIoControl(*hDevice*, **IOCTL_SERIAL_GET_SYNC_STATE**, **NULL**, **0**,
frameFormatBuffer, **sizeof(MCX_SYNCHRONIZATION_PARAMETERS)**,
lpcbBytesReturned, *lpoOverlapped* **)**

HANDLE *hDevice*; /* Handle of the device */
PMCX_SYNCHRONIZATION_PARAMETERS *frameFormatBuffer*; /* pointer to parameters */
LPDWORD *lpcbBytesReturned*; /* size of returned params */
LPOVERLAPPED *lpoOverlapped*; /* overlapped struct. addr */

The SET function can be used to select the frame format : HDLC, etc. The integer pointed to by *lpcbBytesReturned* always takes the value 0. **WARNING** : in driver versions earlier than 1.8.3, this function returns an error if the state of the port set previously by SetCommState contains options not supported by the board (e.g. *ByteSize*=7 in HDLC mode). In the later versions, the port is forced to a "reasonable" state.

The GET function consults the current frame parameters. The integer designated by *lpcbBytesReturned* always takes the value sizeof**(MCX_SYNCHRONIZATION_PARAMETERS)**.

The **MCX_SYNCHRONIZATION_PARAMETERS** structure is made up of the following items :

UCHAR *SynchronousMode*;
Protocol: **MCX_SYNC_HDLC**, **MCX_SYNC_BISYNC**, **MCX_SYNC_LAPB** or **MCX_SYNC_CHAR**. The **MCX_SYNC_CHAR** mode corresponds to asynchronous transmissions. LAPB is also known as HDLC/ABM.

UCHAR *Duplex*;
Simultaneous transmission : **MCX_WAY_FULLDUPLEX** (simultaneous transmission and reception), or **MCX_WAY_HALFDUPLEX** (alternating transmission and reception ; see the details of this mode in the Multiprotocol Manual [DTUS016]).

Important : most of the functionalities used in half-duplex mode can be activated by correctly configuring the full-duplex mode. Half-duplex mode should only be used when it is really necessary (it is the only way to prevent transmission when reception is in progress).

Half-duplex mode forces *fOutxCtsFlow* = TRUE, *fRtsControl* = RTS_CONTROL_TOGGLE, and ignores data received without DCD. It prohibits transmission when DCD is active and also when DSR is active with *fDsrSensitivity* = TRUE.

UCHAR *TransmitClockSource*; Transmit clock source (see below).

UCHAR *ReceiveClockSource*; Receive clock source (see below).

Constant	clock source
MCX_CLOCK_RXC	pin 17 (RxClock)
MCX_CLOCK_TXCI	pin 15 (TxClock)
MCX_CLOCK_BRG	internal bauds generator
MCX_CLOCK_DPLL	decoded in data (only with FM or Manchester coding)

UCHAR *TxClockPinSource*; Clock source available on pin 24 (pin 15 on MCXBP rev A). Warning : this pin is disabled if **MCX_CLOCK_TXCI** is used (by *TransmitClockSource* or *ReceiveClockSource*).

Constant	clock source
MCX_CLOCK_TRXC_HIGH	none (pin set to MARK state)
MCX_CLOCK_TXCLOCK	like *TransmitClockSource*
MCX_CLOCK_BRG	internal bauds generator
MCX_CLOCK_DPLL	decoded in data (only with FM or Manchester coding)

UCHAR *MonosyncChar*; First sync character in BISYNC mode. The only sync character in MONOSYNC mode. Ignored in the other modes.

UCHAR *BisyncChar*; Second sync character in BISYNC mode. Ignored in the other modes.

UCHAR *Version*; Version of the structure. Should always = 1.

USHORT *Options*; Protocol options. Each option is a bit that must be added if the option is to be used.

- **MCX_HDLC_USERDTR** enables the use of the CCITT 108 (DTR) circuit in HDLC, LAPB, and X25 modes on channels 1, 2, and 3 of the boards with MCXBP(MR) or Lite/S or PCB/S extension. However, this will limit the performance levels[1].
- **MCX_HDLC_SPECS** forces the use of the elements of the *Protocol.Hdlc* structure, which are ignored otherwise.
- **MCX_BISYNC_SPECS** forces the use of the elements of the *Protocol.Bisync* structure, which are ignored otherwise.

USHORT *DataLength*; Maximum frame length. If the value is set to 0, the default length will be used (see the PROTO command in the Multiprotocol documentation [DTUS016]).

union {...} *Protocol*; The sub-structures specified here can be used to specify the parameters for a specific protocol.

[1] See the PROTO documentation in the “Multiprotocol Software User Manual” [DT003]

The following should be specified for LAPB only :

UCHAR *Protocol.Lapb.Role*; **MCX_ROLE_CLIENT** if the application acts as a client (ETTD), **MCX_ROLE_NETWORK** if the application serves the network.

UCHAR *Protocol.Lapb.K,N2,T1,T2,T3*
Standardised LAPB parameters. **MCX_LAPB_DEFAULT** will invoke the default value [2].

USHORT *Protocol.Lapb.Spare*; Zone reserved for LAPB.

The following elements should be specified for HDLC only :

UCHAR *Protocol.Hdlc.RxFrames*;
Number of frames acceptable in reception mode without risk of loss if the PC does not read them immediately from the board (number of frame receive buffers on the board). The value 0 resolves to the default value [2].

UCHAR *Protocol.Hdlc.TxFrames*;
Number of frames that the board can memorise as awaiting transmission (number of frame transmit buffers on the board). The **WriteFile()** function will never halt processing if a frame buffer is available on the board at the time of the call. The value 0 resolves to the default value [2].

USHORT *Protocol.Hdlc.Spare1*; à *Spare5;*
Zones reserved for HDLC.

The following elements should be specified for BISYNC only :

UCHAR *Protocol.Bisync.RxFrames*;
Number of frames acceptable in reception mode without risk of loss if the PC does not read them immediately from the board (number of frame receive buffers on the board). The value 0 resolves to the default value [2].

UCHAR *Protocol.Bisync.TxFrames*;
Number of frames that the board can memorise as awaiting transmission (number of frame transmit buffers on the board). The **WriteFile()** function will never halt processing if a frame buffer is available on the board at the time of the call. The value 0 resolves to the default value [2].

USHORT *Protocol.Bisync.Spare1* à *Spare5 ;*
Zones reserved for BISYNC.

[2] See the PROTO command documentation in the « Multiprotocol firmware user's manual » [DTUS016]

II.3 Example for SET_SYNC_STATE

```
#include <windows.h>
#include <winioctl.h>
#include <mcc_mcx.h>

Proto(HANDLE chan, int fonc)
{
     MCX_SYNCHRONIZATION_PARAMETERS sp;
     DWORD count;
     DWORD speed;

     //
     // switch to HDLC mode with the appropriate clocks for a
     // NULL-MODEM cable (internal clock for Tx, external for Rx)
     // with 1024 Data bytes max. per frame
     //
     sp.SynchronousMode = MCX_SYNC_HDLC;
     sp.Version = 1;
     sp.Duplex = MCX_WAY_FULLDUPLEX;
     sp.Options = 0;
     sp.DataLength = 0;  /* default = 1,024 bytes */
     sp.TransmitClockSource = MCX_CLOCK_BRG;
     sp.ReceiveClockSource = MCX_CLOCK_RXC;
     sp.TxClockPinSource = MCX_CLOCK_BRG;

     if(!DeviceIoControl(
               chan,IOCTL_SERIAL_SET_SYNC_STATE,
               &sp,sizeof(sp),NULL,0,&count,NULL)) {
          printf("SET_SYNC_STATE Ioctl: error %d\n",
     GetLastError());
          exit(1);
     }
}
```

II.4 CMD and CMD_AUTO functions

Two communication functions have been added to enable manual dialogue with the on-board interpreter. For a description of the commands, their parameters and data zone, consult the appropriate manual for the "firmware" you are using (basic software [DTUS014] or multiprotocol software [DTUS016]).

#include "windows.h"
#include "winioctl.h"
#include "mcc_mcx.h"

DeviceIoControl(*hDevice*, **IOCTL_SERIAL_CMD**,
paramsFromAppToBoard, *paramsToSize*,
paramsFromBoardToApp, *paramsFromSize*, *lpcbBytesReturned*,
lpoOverlapped
)

DeviceIoControl(*hDevice*, **IOCTL_SERIAL_CMD_AUTO**,
paramsFromAppToBoard, *paramsAppSize*,
paramsFromBoardToApp, *paramsBoardSize*, *lpcbBytesReturned*,
lpoOverlapped
)

HANDLE *hDevice*;	/* Handle of the device */
PMCC_CMD *paramsFromAppToBoard*;	/* pointer to sent parameters */
DWORD *paramsAppSize*,	/* size of sent params */
PMCC_CMD *paramsFromBoardToApp*;	/* pointer to returned parameters */
DWORD *paramsBoardSize*,	/* size of space for returned params */
LPDWORD *lpcbBytesReturned*;	/* size of returned params */
LPOVERLAPPED *lpoOverlapped*;	/* overlapped struct. addr */

The **IOCTL_SERIAL_CMD** function can make the board execute any command.

The **IOCTL_SERIAL_CMD_AUTO** function can make the board execute any command with parameter 1 being initialised by the driver, and the channel number corresponding to hDevice.

The **MCC_CMD** type is a structure that matches that of the board's mailbox. This enables the application to transmit and receive parameters and data.

When a command is sent to the board by these functions, the driver executes one of the following actions :

1) If it exists, the *paramsFromAppToBoard→Data* table is copied into the DATA zone of the board's mailbox,
2) the *paramsFromAppToBoard→Cb.par*[]table is copied into the mailbox's PARAMETERS zone,
3) *paramsFromAppToBoard→Cb.opcode* is copied into the mailbox's OPCODE zone,
4) the binary value 0000 0001 is written into the mailbox's VALIDATION byte. This causes the board to execute the command. The board then issues an end of command interrupt that enables the driver to continue processing,
5) the mailbox's STATUS zone is copied into *paramsFromBoardToApp→Cb.status*

6) the PARAMETERS zone in the mailbox is copied into *ParamsFromBoardToApp→ Cb.par*[],
7) if the *paramsFromAppToBoard→Data* existe, table exists, the mailbox's DATA zone is copied into *paramsFromBoardToApp→Data*,
8) the application is woken up or alerted depending on *the lpoOverlapped value*.

The length of the **MCC_CMD** *Data* field is variable. For example, the structure can be created by dynamically allocating (sizeof(**MCC_CMD**)+data_length) bytes; the **MCX_DIRECT_IO_BUFFER_SIZE**(*datalen*) macro can be used to calculate the required number of bytes. The driver recognises the existence of the *Data* field if the *Cb.length* field is non-null. Otherwise, it assumes that the command that is to be executed does not use the mailbox's DATA zone.

paramsAppSize must be equal to the sum of sizeof(**MCC_CMD**) and the length of the data zone.

If the command has been executed correctly, the integer designated by *lpcbBytesReturned* will still equal *paramsBoardSize*.

The **_MCC_CMD** structure contains the following elements :

unsigned char *Cb.opcode*;	Code of the command that is to be executed. **mcc_mcx.h** defines the symbolic names for these codes.
unsigned char *Cb.status*;	The returned result supplied by the board in the STATUS zone.
unsigned char *Cb.par[76]*;	The command parameters; returned parameters for some commands.
unsigned char *Cb.ichan*;	}
unsigned char *Cb.icond*;	} Returns a copy of the mailbox INTERRUPT
unsigned char *Cb.ipar1*;	} zone. Theoretically, these five elements are not
unsigned char *Cb.ipar2*;	} used[3].
unsigned char *Cb.ipar3*;	}
unsigned char * *Cb.data*;	Unused in Windows NT[4].
unsigned char * *Cb.kdata*;	Field used temporarily by the driver during command execution.
unsigned short *Cb.length*;	Working length of the data zone in bytes.
unsigned char *Data*[0...];	Data zone that will be exchanged with the board's mailbox; because this zone must immediately follow the *Cb* structure, the **MCC_CMD** variable length structure is used[5].

[3] except for the LDIAL command used by the MCC board.
[4] In a UNIX environment, pointer to the data zone that will be exchanged with the board's mailbox.
[5] This zone is not used in the UNIX driver.

Notes

- ☞ To avoid errors, it is advisable to include the same pointer in *paramsFromAppToBoard* and *paramsFromBoardToApp*, and the same length in *paramsAppSize* and *paramsBoardSize*.
- ☞ To ensure compatibility with the UNIX driver, we suggest initialising *paramsFromAppToBoard→Cb.data = paramsFromAppToBoard→Data*.
- ☞ To avoid confusion between *Cb.par*[]which starts at *Cb.par*[0], and the command description which starts at PARAMETER 1, it is advisable to reference PARAMETER N by :
  paramsFromAppToBoard→Cb.par[**N-1**] to quote the PARAMETRE **N**.

Interactions between these functions and normal driver operation

The description below is up-to-date for version 1.8.3 of the driver. These interactions may change in future versions.

ALLOC :

The driver detects this command and, if necessary, adjusts the buffer size information. It can consequently replace SetupComm() which is ignored.

CHDEF :

A CHDEF is executed by the driver during SetCommState, if an attempt is made to change DCB.EvtChar. In this case, only EvtChar is included in the CHDEF command.

MINTR :

- This command is executed during CreateFile, CloseHandle, DeviceIoControl (IOCTL_SERIAL_SET_SYNC_STATE), SetCommMask (if the EV_RXFLAG is activated or deactivated) and in some cases during ReadFile.
- The activated sources are: IT1, IT2 (bit Mde=1), IT3 (if EV_RXFLAG is active), IT5, IT6, IT7.

PROTO :

- The driver detects this command and adjusts its own protocol information. It can consequently replace DeviceIoControl SET_SYNC_STATE to enable the use of non-standardised parameters ; this does not disrupt driver operation.
- The VINIT, RXENB, MINTR and, in some cases VMODE, EscapeCommFunction() commands must then be executed to position RTS and DTR, and PurgeComm() to purge the buffers.

RSMDE :

- No undesirable interaction.

II.5 Examples for CMD and CMD_AUTO

These examples can be adapted to send any command appearing in the firmwares manuals.

SetElectricalInterface() switches an MCX serial channel to RS232, RS422, EIA530… by sending the RSMDE command to the board's firmware.

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
#include <winioctl.h>
#include <mcc_mcx.h>
...
//
// For other available electrical interface codes, see the
// Multiprotocol firmware documentation (DTUS016), in the
// RSMDE command section.
//
#define RSMDE_RS232      0
#define RSMDE_RS422      1
#define RSMDE_EIA530     6

BOOLEAN SetElectricalInterface(HANDLE hfd)
{
     MCC_CMD cmd;
     long return_bytes;
     int lasterror;

     //
     // init opcode and parameters, see RSMDE command in
     // basic software or multiprotocol documentation
     //
     cmd.Cb.opcode = RSMDE;
     cmd.Cb.par[1-1] = 0; // Will be replaced by channel n°
     cmd.Cb.par[2-1] = RSMDE_RS422; // Set RS422 interface
     // Change RSMDE_RS422 to the value adequate for
     // your application
     cmd.Cb.length = 0;   // No data zone required
     // Adapt the RELRP example when a data zone must be used

     if (!DeviceIoControl( hfd, IOCTL_SERIAL_CMD_AUTO,
               &cmd,sizeof(cmd),
               &cmd,sizeof(cmd),
               &return_bytes,NULL)) {
          printf("SetElectricalInterface: error, code %d\n",
                    GetLastError());
          return FALSE;
     }
     if (command->Cb.status != 0) {
          printf("RSMDE: failed, status %d\n",
               command->Cb.status);
          return FALSE;
     }
     return TRUE;
}
```

Relrp() runs the RELRP command on the board. This command is not linked to a particular channel. The command sends information on the board type and capabilities.

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
#include <winioctl.h>
#include <mcc_mcx.h>

BOOL Relrp()
{
   PMCC_CMD command;/* space for RELRP and its parameters */
   DWORD retLen;    /* length returned by DeviceIoControl */
   int cmdLen;      /* length of command structure */
   int dataLen;     /* Length of the RELRP Data area */
   HANDLE hDevice;

   // one of the channels must be used (any one will do)
   hDevice = CreateFile("\\\\.\\COM3",
               GENERIC_WRITE|GENERIC_READ, 0, NULL,
               OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);

   //
   // init opcode and data, see RELRP command in
   // basic software or multiprotocol documentation
   //
   dataLen = 18;
   cmdLen = MCX_DIRECT_IO_BUFFER_SIZE(dataLen);
   command = malloc(cmdLen);
   command->Cb.opcode = RELRP;
   command->Cb.length = dataLen;
   // the following assignment distinguishes the MCC boards
   // which do not modify data item 18.
   command->Data[18-1] = MCX_TYPE_MCC;

   if (!DeviceIoControl(hDevice,IOCTL_SERIAL_CMD,
        command, cmdLen, command, cmdLen, &retLen, NULL)) {
      printf("RELRP: Win32 error %d\n", GetLastError());
      return FALSE;
   }
   if (command->Cb.status != 0) {
      printf("RELRP: failed, status %d\n", command->Cb.status);
      return FALSE;
   }
   printf("RELRP: carte type %d à %d MHz, %d ports\n",
        command->Data[18-1], command->Data[11-1],
        command->Data[9-1]);

   free(command);
   CloseHandle(hDevice);
   return TRUE;
}
```

II.6 ACCESS_AREA function

This function allow to read and write directly in the board mailbox.

#include "windows.h"
#include "winioctl.h"
#include "mcc_mcx.h"

DeviceIoControl(*hDevice*, **IOCTL_SERIAL_ACCESS_AREA**,
pAccessDescriptor, accessDescriptorLength,
pGetBuffer, getBufferLength, *returnedLength*,
lpoOverlapped)

HANDLE *hDevice*; /* Handle of the device */
PMCX_AREA_DESCRIPTOR *pAccessDescriptor*; /* pointer to operation and its parameters */
DWORD *accessDescriptorLength*; /* size of operation and its parameters */
PVOID *pGetBuffer*; /* pointer to returned data, NULL if unused */
DWORD *getBufferLength*; /* size of buffer for return data, 0 if unused */
LPDWORD *returnedLength*; /* pointer to actual size of returned data */
LPOVERLAPPED *lpoOverlapped*; /* overlapped structure address */

This function allow to read or modify board mailbox content. The structure pointed to by *pAccessDescriptor*, of length accessDescriptorLength, is transmitted to the driver. The driver processes the operation. It can return data in the structure pointed to by pGetBuffer of length *getBufferLength*, and it indicates the length of returned data in the word pointed to by *returnedLength. The informations returned are truncated to the shortest size indicated by getBufferLength* and **returnedLength*.

The **_MCX_AREA_DESCRIPTOR** structure is composed of the following elements :

long *Operation*; Operation code (see below)
long *StartAddress*; Address, relative to the beginning of the mailbox board selected by *hDevice*, where will take place writing or reading.
long *Length*; Number of bytes to transfer
unsigned char *Buffer*[...]; Data to write in the mailbox (use by MCX_AREA_SET operation).

Buffer[] contains only one byte. To write several characters, the following macro allows to allocate a structure of the required size:

MCX_AREA_DESCRIPTOR_SIZE(datalen).

Recognized operations are ::

MCX_AREA_GET+MCX_AREA_MEMORY Mailbox contents starting at address *StartAddress* for a length of *Length* are copied to the place pointed to by *pGetBuffer* with a maximum length of getBufferLength. **returnedLength* is loaded with the length actually copied.

MCX_AREA_SET+MCX_AREA_MEMORY *Buffer* contents are copied to address *StartAddress*in the mailbox for a length of *Length* bytes.

II.7 MCX_OPTIONS function

A function was added to access miscellaneous parameters or to activate specific behaviours of the boards.

#include "windows.h"
#include "winioctl.h"
#include "mcc_mcx.h"

DeviceIoControl(*hDevice*, **IOCTL_SERIAL_MCX_OPTIONS**,
pOptionDescriptor, *optionDescriptorLength*, *pResultsBuffer,*
resultsBufferLength, *returnedResultsLength*, *lpoOverlapped*)

HANDLE *hDevice*; /* Handle of the device */
PMCX_OPTION *pOptionDescriptor*; /* pointer to option code and parameters */
DWORD *optionDescriptorLength*; /* size of option code and parameters */
PMCX_OPTION *pResultsBuffer*; /* pointer to returned data, NULL if unused */
DWORD *resultsBufferLength*; /* size of buffer for return data, 0 if unused */
LPDWORD *returnedResultsLength*; /* pointer to actual size of returned data */
LPOVERLAPPED *lpoOverlapped*; /* overlapped structure address */

This function allows modification of some specific parameters of the driver and board. The structure pointed by *pOptionDescriptor*, of length *optionDescriptorLength*, is transmitted to the driver. The driver processes the option. It can return data in the structure pointed by *pResultsBuffer of lentgh resultsBufferLength*, and it indicates returned information length in the word pointed to by *returnedResultsLength.* The returned information may have been truncated to the size indicated by *resultsBufferLength*.

The **_MCX_OPTIONS** structure is composed of the following elements.

long *Option*; Option code ; only useful in the transmitted structure.
union{long *Long***; ...}** *Value*; Parameters/data returned by option. Their type (**long**, **short** or **unsigned** char) depends on the requested option. Depending on the expected type, the various union elements (*Long, Short, Char*) will be used.

If an option requires several parameters, the macro MCX_OPTION_SIZE(paramslen) allows to compute the required size in order to dynamically allocate the structure.

Options recognized in this version of the driver are :

MCX_OPTION_GET_CHANNEL

Ask for the number of the channel associated to *hDevice*. Returned structure contains a **short** (*pOptionDescriptor→ Value.Short[0])* holding the channel number, relative to its parent board. For example, if two MCX boards with sixteen channels are setup with names COM3…COM34, the returned number for the COM3 and COM19 handles will be 1, the returned number by COM 18 and COM34 will be 16.

Additional information will be returned in future versions, hence **returnedResultsLength* might be greater than *resultsBufferLength.*

MCX_OPTION_GET_RSMODE

The *pOptionDescriptor→Value.Char[0]* element of the returned structure contains the code of the electrical interface currently set for this port. The possible values are indicated in the relevant firmware manual.

MCX_OPTION_SET_DSR_RI_INVERSION

pOptionDescriptor→Value.Char[0] element of provided structure must contain TRUE or FALSE. If it contains TRUE, the processing of the DSR and RING signals is inverted on the channel.

MCX_OPTION_GET_DSR_RI_INVERSION

pOptionDescriptor→Value.Char[0] element of the returned structure contains TRUE if DSR and RING signals processing is inverted on the channel, else FALSE.

MCX_OPTION_GET_COMPATIBILITY

pOptionDescriptor→Value.Long[0] element of the returned structure contains the Compatibility variable value as set for the channel. This variable reflects driver configuration options, may be modified by MCX_OPTIONS function. returned bits values should be analyzed with the following definitions available in "mcc_mcx.h" :

MCX_COMPAT_LOWSIGS	– RTS/DTR down at 1st CreateFile
MCX_COMPAT_DSR_RI	– DSR & RING inversion
MCX_COMPAT_DTRFLOW	– compat.1.6.2(EPROM 1.8/3.8)
MCX_COMPAT_V164	– V1.6.4 compatibility
MCX_COMPAT_V171	– V1.7.1 compatibility
MCX_COMPAT_V181	– V1.8.1 compatibility
MCX_COMPAT_QSIZE	– V1.8.2 compatibility

Consult the installation section for more details.

MCX_OPTION_GET_PHYSICAL_PARMS

Request physical board parameters. Returned structure contains MCX_OPTION_PHYSICAL_PARMS, which is a substructure aligned with *pOptionDescriptor→Value.Long[0]*. Defined elements depend one the board and bus type. For PCI/cPCI boards, the following physicals resources elements are returned :

StructVersion	0
BusType;	5 (DDK's "PCIBus" macro)
BusNumber;	PCI bus number where board is.
SlotNumber;	Board place on bus
MemoryAddressLow;	Mailbox address
MemoryAddressHigh;	High bit address (64 bits bus)
MemorySize;	32768
IoAddressLow;	Board I/O port address
IoAddressHigh;	High bit address (64 bits bus)
IoSize;	8
InterruptVector;	Interruption
DmaChannel;	-1
SpecificInformation;	0

A programming example is on the distribution medium in the file "\sdk\multiprotocol\examples\common\special\getphys.c"

II.8 Examples for ACCESS_AREA and MCX_OPTIONS

```
#include <windows.h>
#include <winioctl.h>
#include <mcc_mcx.h>

DemoOptions(HANDLE hDevice)
{
     DWORD retLen;  /* returned length in calls */
     int canal;     /* physical channel, GET_CHANNEL result */


     { /* get channel number */

          PMCX_OPTION pOptionDesc; /* example with  malloc */
          MCX_OPTION optionResult; /* example without malloc */

          pOptionDesc = malloc(MCX_OPTION_SIZE(0));
          pOptionDesc->Option = MCX_OPTION_GET_CHANNEL;
          DeviceIoControl(hDevice,IOCTL_SERIAL_MCX_OPTIONS,
                         pOptionDesc, MCX_OPTION_SIZE(0),
                         &optionResult, sizeof(optionResult),
                         &retLen, NULL);
          free(pOptionDesc);
          canal = optionResult.Value.Short[0];
     }

     /* the following line loads the board buffer */
     WriteFile(hDevice, “ABC”, 3, &retLen, NULL);
     /* there is no Overlap struct, so BTRAN is */
     /* over when WriteFile finishes */

     { /* obtain the emission counter with ACCESS_AREA */
       /* notice : equivalent result is more easily obtained */
       /* with ClearCommError(Win32) function */

          MCX_AREA_DESCRIPTOR areaDesc;
          USHORT txCount;

          areaDesc.Operation = MCX_AREA_GET+MCX_AREA_MEMORY;
          areaDesc.StartAddress = 0x7f80 + 2*(canal-1);
          areaDesc.Length = sizeof(txCount);
          DeviceIoControl(hDevice,IOCTL_SERIAL_ACCESS_AREA,
                         &areaDesc, sizeof(areaDesc),
                         &txCount, sizeof(txCount),
                         &retLen, NULL);
          printf("Place libre dans le tampon d’émission : ");
          printf("%u\n",txCount);
     }
}
```

```
#include <windows.h>
#include <winioctl.h>
#include <mcc_mcx.h>

DemoPhysical(HANDLE hDevice)
{
      DWORD retLen;         /* returned length in calls */
      PMCX_OPTION pOptionDesc;
      PMCX_OPTION_PHYSICAL_PARMS pOptionResult;
      int optionLen;

      optionLen = MCX_OPTION_SIZE(sizeof(MCX_OPTION_PHYSICAL_PARMS));
      pOptionDesc = malloc(optionLen);

      pOptionDesc->Option = MCX_OPTION_GET_PHYSICAL_PARMS;
      DeviceIoControl(hDevice,IOCTL_SERIAL_MCX_OPTIONS,
            pOptionDesc, optionLen,    // provided parameters
            pOptionDesc, optionLen,    // returned informations
            &retLen, NULL);

      pOptionResult = (PMCX_OPTION_PHYSICAL_PARMS)
                           pOptionDesc->Value.Long ;
      printf("Adresse physique de la carte : %x%x / %x%x / %d\n",
            pOptionResult->MemoryAddressHigh,
            pOptionResult->MemoryAddressLow,
            pOptionResult->IoAddressHigh,
            pOptionResult->IoAddressLow,
            pOptionResult->InterruptVector);

      free(pOptionDesc);
}
```

II.9 Appendix : flow control.

The boards in the MCX range, equipped with the MCX-MULTIPROTOCOL option, and used with a version of the driver later than 1.7.0, **support all the flow controls offered by the Win32 API**, and even some additional controls (accessible by executing the VMODE command directly).

Les cartes équipées du logiciel de base, ou de l'option Multiprotocole utilisée avec un pilote antérieur à la version 1.7.1, ne peuvent pas supporter tous les contrôles de flux proposés dans l'API de Win32. The restrictions affecting the operation of flow control are described below :

Methods supported

The following flow control options are supported :

- none
- XON/XOFF configurable
- DTR/CTS
- RTS/CTS since «firmwares» rév. 2.0 (MCX) and 3.8 (MCC)

In all cases, both transmission directions are controlled. The two directions cannot be configured independently.

Configuring the control method

The *SetCommState* service and the interpretation of the fields in the *DCB* structure (see page 27) have been adapted to address these constraints. The use of the Multiprotocol option and the board configuration options (i.e. the Register *Synchronous* and *Compatibility* values) also affect operation. Flow control is consequently supported as follows :

a) if the *Synchronous* indicator is set to 1 (with an old driver and/or old firmware), flow control will not be supported because the board does not support the VMODE command,
b) otherwise, if *fInX* or *fOutX* is TRUE, flow control is XON/XOFF with the *XonChar* and *XoffChar* characters,
c) otherwise, if *fRtsControl* is set to RTS_CONTROL_HANDSHAKE, flow control is implemented by hardware with the RTS and CTS signals,
d) otherwise, if *fOutxCtsFlow* is TRUE or *fOutxDsrFlow* is TRUE or if *fDtrControl* is set to DTR_CONTROL_HANDSHAKE, flow control is implemented by hardware with the DTR and CTS signals,
e) otherwise there is no flow control.

Compatibility with earlier versions

Earlier versions of the boards and software did not support flow control by RTS/CTS. In any of the following cases :

- the driver version is 1.6.2 or earlier,
- the "EPROM 1.8/3.8" configuration option has been validated,
- the version of the board firmware is earlier than 1.8,

Rules c) and d) must be combined into a single rule :

c+d) otherwise, if *fOutxCtsFlow* is TRUE or *fOutxDsrFlow* is TRUE or *fDtrControl* is set to DTR_CONTROL_HANDSHAKE or *fRtsControl* is set to RTS_CONTROL_ HANDSHAKE, flow control is implemented by hardware with the DTR and CTS signals,

Note

To ensure compatibility with future versions, use the combination that matches the cable you are actually using (e.g. if the incoming control signal is on CTS, use *fOutxDsrFlow* to manage it instead of *fOutxCtsFlow*).

III APPENDIX : SPECIFIC ERROR CODES.

The error codes returned by the Win32 API are as described in the API documentation. Here, however, is a selected list of error codes that are not easily understood :

2 **ERROR_FILE_NOT_FOUND**

The driver has not started. Consult the Event Viewer and the Device Manager.

5 **ERROR_ACCESS_DENIED**

The channel has already been opened by another process.

21 **ERROR_NOT_READY**

In **WriteFile**, transmission is impossible because the Data Link layer (LAPB) is not connected.

23 **ERROR_CRC**

In a **ReadFile**, this error indicates that the received frame contains an error (all types of error generate this error code, and not just CRC errors).

38 **ERROR_HANDLE_EOF**

In LAPB protocol, indicates that the Data Link layer is disconnected before or after the execution of a **ReadFile**, and that there is no outstanding frame in the receive buffer.

57 **ERROR_ADAP_HDW_ERR**

Unexpected error on the MINTR command. Probably a board malfunction.

87 **ERROR_INVALID_PARAMETER**

1) In **DeviceIoControl**, **WaitCommEvent**, **ReadFile** and **WriteFile**, this error can, in particular, indicate that the *lpoOverlapped* parameter does not match the options requested in the **CreateFile**; either that or the hEvent element in the OVERLAP structure is incorrect.
2) In **DeviceIoControl**, either one of the parameters or one of the elements in the structure supplied in the third parameter position is incorrect.

122 **ERROR_INSUFFICIENT_BUFFER**

1) The length specified in **DeviceIoControl** is wrong.
2) In **DeviceIoControl**, the value of the *Cb.length* element in the IOCTL_SERIAL_CMD or CMD_AUTO function is too small.

995 **ERROR_OPERATION_ABORTED**

1) The board did not answer a command within the timeout. The most likely causes are : the interrupt supplying the active IRQ is not pushed in or is the wrong one ; the board is very busy, in which case the CommandTimeout parameter value should be increased ; or there is a constant influx of parasites on the channel.
2) In LAPB protocol, a **WriteFile** was attempted when the link was down.

997 **ERROR_IO_PENDING**

The operation has not been completed. See **GetOverlappedResult()**.

1450 **ERROR_NO_SYSTEM_RESOURCES**

A **DeviceIoControl** sent to the driver has not been recognised. The likely causes are : the application sends a **DeviceIoControl** with a bad code ; the version of Windows NT is not supported.

1784 **ERROR_INVALID_USER_BUFFER**

In **WriteFile**, the 'count' parameter is too high for the size of the buffer or the frames, or exceeds the 31 Kbyte limit.

IV APPENDIX : LIMITATION AND DIFFERENCES WITH THE COM PORTS

Several restrictions are due to the interactions between the different capabilities of the Win32 API, the driver, the firmware, and the board itself.

The board's basic software limits the available flow control types (see Appendix entitled "Flow control" above). The Multiprotocol option, though less effective in synchronous mode, is not affected by this restriction.

134.5 bauds transmission speed : the DCB does not support this speed because BaudRate is a LONG variable. However, the driver does support this speed via the value (ULONG)(-134).

The DSR signal (circuit 107) does not exist on all boards (see the board connector documentation). A configuration option and an API function can be used to swap the processing of this signal with RING, which enables a pseudo-DSR implementation, if an appropriate cable is used.

In synchronous mode, the RING signal (circuit 125) is only available as from release E of the MCXBP connection box.

DTR in synchronous mode : see the Multiprotocol software manual [DTUS016] and the MCX_HDLC_USERDTR indicator on page 40.

Supported speeds : see the appropriate firmware manual [DTUS014], [DTUS016].

V COM PORTS FAQ

Q : My program executes a WriteFile which does not return an error, but the data are not completely sent out.

R : You can use one of the following functions to wait for the end of transmission : PurgeComm(), FlushFileBuffers(), CloseHandle() or a program exit.

Q : My program executes a WriteFile which does not return an error, but no data is not sent out.

R : see previous question. Also, check that the transmission clock exists and is correctly programed (with the PROTO command or the DeviceIoControl `IOCTL_SERIAL_SET_SYNC_STATE`).

Q : I cannot see the changes on the DSR pin.

R : DSR is not handled by the « Basic firmware » nor by the « Multiprotocol firmware ».
You can use the advanced configuration tab of the board to overcome this problem.

MCXDOS/AUTOMCX MODE

The following sections describes « Mcxdos/Automcx » and « BIOS extension » driver operating modes.

Mcxdos/Automcx mode allow to load an application on board, in a DOS environment.

I DEVELOPMENT OF THE APPLICATION TO BE DOWNLOADED

Applications which will run on board in this mode, must be first created and tested on a development computer using MS-DOS, Windows 9x or compatible. It can be created by any native- or cross-development tool provided it generates compatible DOS executable code. Thus, MSC 6 and 7, MSVC 1.52, Borland C, Pharlap environments were used successfully.

The development phase needs the MCXDOS tool which allows to share the host computer peripherals with the MCX board, then the creation of a « BOOTFILE » which is a diskette image contained in a disk file of corresponding size. This « BOOTFILE » contains the DOS system and the application to download.

II PROVIDED UTILITIES

Four utilities appear on the distribution medium in the SDK\AUTOMCX\EXE directory :

AUTOMCX	program loading « BOOTFILE » to board's memory.
RESETMCX	hot restart program for a PCI/cPCI board.
MCXIO	utility to execute inputs-outputs on the board's memory and IO ports board.
MCXDEBUG	program allowing display and modification of the dual ported memory of the board.

III BOARD LOADING

Run AUTOMCX in command line with two parameters : « name prefix » of the board typed during setup, and BOOTFILE file name. For example :

```
automcx  mcx0403  MCXBOOT
```

IV LOW LEVEL PROGRAMMING INTERFACE

Win32 « CreateFile », « DeviceIoControl » « CloseHandle » functions allow access to a configured card in automcx mode as follows:

```
#include <winioctl.h>
#include <mcc_mcx.h>

void *Mailbox;  /* Replace 'void' with a structure appropriate
                for the application.*/
char *nom_carte[] = « \\\\.\\mcx_exemple »;
DWORD cbReturned;
MCX_AREA_DESCRIPTOR Area;  /* transfer parameters for I/O ports */

/* open and close board access*/
HANDLE Handle = CreateFile(nom_carte,
                     GENERIC_READ | GENERIC_WRITE,
                     0, NULL, OPEN_EXISTING,
                     FILE_ATTRIBUTE_NORMAL, NULL);
CloseHandle(Handle) ;

/* obtain a pointer on shared memory window of the board */
BOOL ok = DeviceIoControl(Handle,
                IOCTL_AUTOMCX_MAP_MAILBOX, NULL, 0,
                &Mailbox, sizeof(PVOID), &cbReturned, 0);

/* free above pointer to free used resources */
BOOL ok = DeviceIoControl(Handle,
                IOCTL_AUTOMCX_UNMAP_MAILBOX, NULL, 0,
                NULL, 0, &cbReturned, 0);

/* write on board's I/O port*/
Area.Operation = MCX_AREA_SET|MCX_AREA_IOPORT;
Area.StartAddress = 0 à 7;      /* relative I/O port number */
Area.Length = 1;
Area.Buffer[0] = (UCHAR)cData;  /* character to write*/
Ok = DeviceIoControl(Handle,
                IOCTL_AUTOMCX_WRITE_PORT, &Area, sizeof(Area),
                NULL, 0, & cbReturned, 0);

/* read board's I/O port */
Area.Operation = MCX_AREA_GET|MCX_AREA_IOPORT;
Area.StartAddress = 0 à 7;      /* relative I/O port number */
Area.Length = 1;
ok = DeviceIoControl(Handle,
                IOCTL_AUTOMCX_READ_PORT, &Area, sizeof(Area),
                &Area, sizeof(Area), &cbReturned, 0);
/* result is available in Area.Buffer[0] */

/* learn board's physical resources */
```

It is possible to use **MCX_OPTION_GET_PHYSICAL_PARMS** option of MCX_OPTIONS documented in section II.7. Refer to this section for more details.

Simplified C language interface (provided as an example)
This interface uses the low level interface to offer more convivial functions. You can include the source in your application , or add the provided library « multiprotocol\lib\automcx.lib » in your project.

Warning, this interface is provided as example : it doesn't pretend provide an evolved tool, or treat all error cases.

```
#include <autont.h>

/* open and close board access */
HANDLE Handle = OpenAutomcx(nom_de_carte) ;                /* le préfixe \\.\ est facultatif */
CloseAutomcx(Handle) ;

/* obtain a pointer on window sharing memory of the board */
void *Mailbox = GetAutomcxPtr(Handle) ;

/* write on I/O port of the board */
McxOutPort(Handle, port, valeur) ;

/* read I/O port of the board */
/* return –1 on error, else (int)(unsignedchar)valeur */
int valeur = McxInPort(Handle, port) ;

/* return the 1st byte of fifo, or '\xff' on error */
unsigned char McxReadFifo(Handle) ;
```

V KNOWN DEFECTS

Date synchronization is not implemented in AUTOMCX.EXE utility.

VI USING MCXDEBUG.EXE

MCXDEBUG is a program similar to the debug tool on DOS or Windows, except it is limited to the 32Kb of the PCI/cPCI board's mailbox.

available functions are following :

- ? : help
- d : dump
- i : input
- o : output
- q : quit
- w : write

DUMP : list 128 Bytes of the mailbox

Two syntaxes are possible :

- d
  If you typed 'd' for the first time , the program list the 128 first bytes of the mailbox.
  If you retyped 'd', it will list the 128 next and so on until the end of the mailbox.

- d *offset*
  Allows to reach directly requested *offset*.

Remarks :

- offset size cannot exceed 2 bytes
  If you entre 'd 555555', the program will interpret the command as if you had enter 'd 5555' (see Figure 1).

- If you try to access to an offset out of the mailbox, you will display its 128 last bytes (see Figure 2).

- If you enter for example 'd 7f50', you will list the 128 bytes from 0x7f50 to 0x7fcf.
  Then, if you typed 'd', you will display the 48 bytes remaining from 0x7fd0 to 0x7fff, like shown on Figure 3.

Figure 1 : a dummy dump example

Figure 2 : a dumpout of bounds

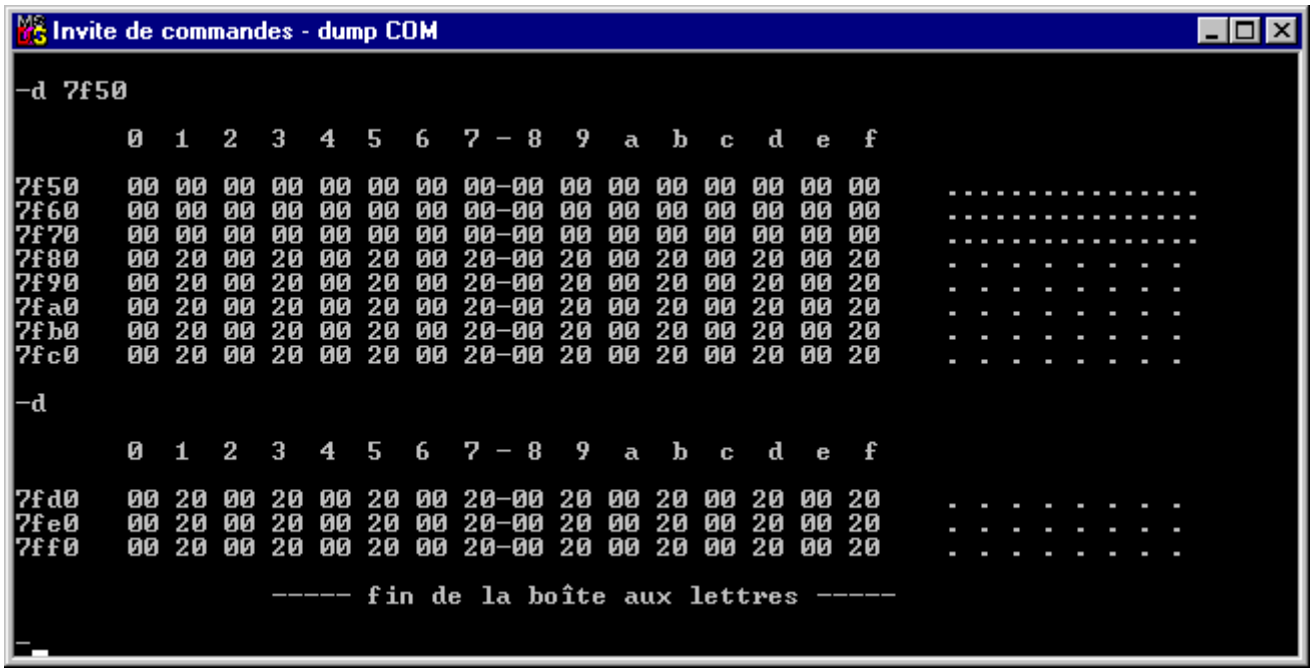


Figure 3 : a dump near the upper bound

INPUT : get a byte from I/O port

Ex : -i [add io]

OUTPUT : send a value on I/O port

Ex : -o [add io] [valeur]

WRITE : write in board's mailbox

There are two ways to write in the mailbox :

- w + offset + text between quotation marks (Figure 4)

- w + offset + hexadecimal value (Figure 5)

Figure 4 : write a text

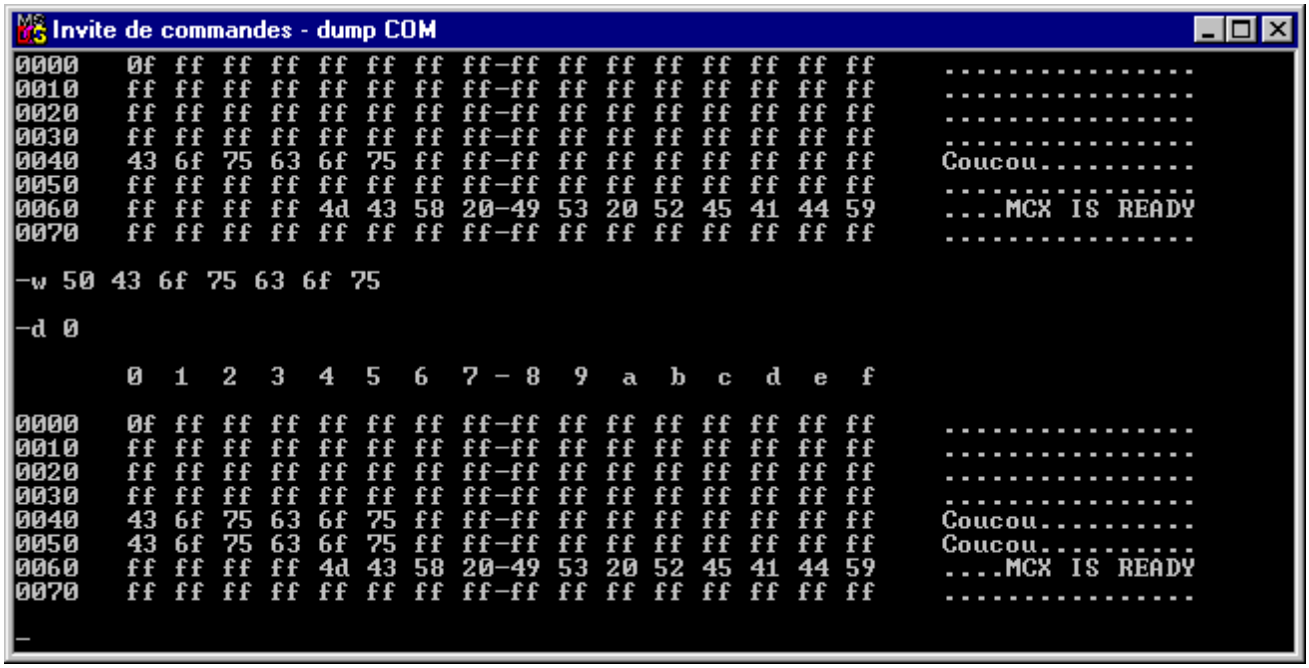


Figure 5 : write directly values in hexadecimal

GLOSSARY

API

Application Programming Interface. The whole of the specifications which allow to an application program to use a subsystem , in our case, MCXDOS or a peripheral driver.

Built-in firmware

Installed in a EPROM on the board, it can be one of : « Logiciel de base », « logiciel multiprotocole » or a custom application software.

Channel, port

Set of elements allowing the transfer of data on one of the connectors on a multichannel board.

Driver, peripheral driver

software provided by ACKSYS, integrated to the operating system, which allows to program and to use multichannel board, independently of materials details (physical address, board driving algorithms ...).

Basic software :

Standard software installed in a EPROM on the board, it allow to use channel of board with MCXBP(MR) or Lite/S or PCB/S extension only, in asynchronous transmition mode only. It is compatible with basic software of MCC boards.

Multiprotocol software :

Installed in surplus of basic software if you had ordered MCX-MULTIPROTOCOLE option, provided in standard on the board with Lite/570 or PCB/570 extension, it allows to use channels in all a range of synchronous and asynchronous formats of transmission. It is partially compatible with basic software.

MCC

The ancestor of the MCX board, supporting 8 or 16 asynchronous channels. To ensure perenniality of applications development by its customers, ACKSYS provides on MCX board range a dialog protocol compatible with that of the MCC : the basic software.

UART

(Universal Asynchronous Receiver/Transmitter) asynchronous send/receive hardware component. Most known are components 8250, 16550…

USART

(Universal Synchronous/Asynchronous Receiver/Transmitter) asynchronous and synchronous send/receive hardware component. MCX boards use components SCC 85C30, or HD64570.